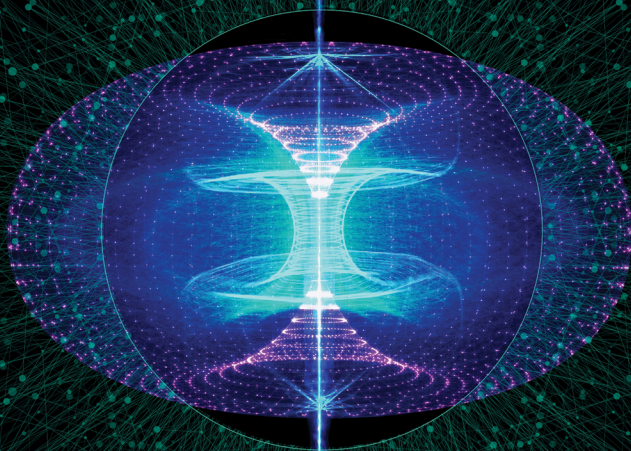


ARTIFICIAL INTELLIGENCE FOR HIGH ENERGY PHYSICS



Editors

Paolo Calafiura · David Rousseau · Kazuhiro Terao

 World Scientific

VISIT...

LANZAROTE
Caliente.COM

ARTIFICIAL
INTELLIGENCE
FOR
HIGH ENERGY
PHYSICS

Other Related Titles from World Scientific

Introduction to High Energy Physics: Particle Physics for the Beginner

by Lee G Pondrom

ISBN: 978-981-122-209-2

ISBN: 978-981-122-301-3 (pbk)

The Adventure of the Large Hadron Collider: From the Big Bang to the Higgs Boson

by Daniel Denegri, Claude Guyot, Andreas Hoecker and Lydia Roos

foreword by Carlo Rubbia, Nobel laureate in Physics 1984

ISBN: 978-981-3236-08-0

ISBN: 978-981-3237-11-7 (pbk)

New Era for CP Asymmetries: Axions and Rare Decays of Hadrons and Leptons

by Ikaros I Bigi, Giulia Ricciardi and Marco Pallavicini

ISBN: 978-981-3233-07-2

Structural Aspects of Quantum Field Theory and Noncommutative Geometry

In 2 Volumes

Second Edition

by Gerhard Grensing

ISBN: 978-981-123-800-0 (Vol. 1)

ISBN: 978-981-123-801-7 (Vol. 2)

ARTIFICIAL INTELLIGENCE FOR HIGH ENERGY PHYSICS

Editors

Paolo Calafiura

Lawrence Berkeley National Laboratory, USA

David Rousseau

Laboratoire de Physique des 2 Infinis Irène Joliot-Curie, France

Kazuhiro Terao

SLAC National Accelerator Laboratory, USA

 **World Scientific**

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI • TOKYO

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

Library of Congress Cataloging-in-Publication Data

Names: Calafiura, Paolo, editor. | Rousseau, David (Physics), editor. | Terao, Kazuhiro, editor.

Title: Artificial intelligence for high energy physics / editors,

Paolo Calafiura, Lawrence Berkeley National Laboratory, USA,

David Rousseau, Laboratoire de Physique des 2 Infinis Irène Joliot-Curie, France,

Kazuhiro Terao, SLAC National Accelerator Laboratory, USA.

Description: New Jersey : World Scientific., [2022] | Includes bibliographical references and index.

Identifiers: LCCN 2021033763 | ISBN 9789811234026 (hardcover)

Subjects: LCSH: Particles (Nuclear physics) | Artificial intelligence.

Classification: LCC QC793.2 .A78 2022 | DDC 539.7/6028563--dc23

LC record available at <https://lcn.loc.gov/2021033763>

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

Copyright © 2022 by World Scientific Publishing Co. Pte. Ltd.

All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system now known or to be invented, without written permission from the publisher.

For photocopying of material in this volume, please pay a copying fee through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA. In this case permission to photocopy is not required from the publisher.

For any available supplementary material, please visit

<https://www.worldscientific.com/worldscibooks/10.1142/12200#t=suppl>

Desk Editor: Ng Kah Fee

Typeset by Stallion Press

Email: enquiries@stallionpress.com

Printed in Singapore

Contents

Chapter 1. Introduction	1
<i>Paolo Calafiura, David Rousseau and Kazuhiro Terao</i>	
1. Artificial Intelligence at the Frontiers of High-Energy Physics	1
2. Why This Book, and Who Should Read It?	2
3. Pre-requisites	3
4. How to Use This Book	3
References	4
Part I: Discriminative Models for Signal/Background Boosting	7
Chapter 2. Boosted Decision Trees	9
<i>Yann Coadou</i>	
1. Introduction	9
2. Specificity of High-Energy Physics	10
3. Decision Trees	20
4. Boosted Decision Trees	34

5. Other Averaging Techniques	53
6. Software	54
7. Conclusion	55
References	55

Chapter 3. Deep Learning from Four Vectors 59

*Pierre Baldi, Peter Sadowski and
Daniel Whiteson*

1. Introduction: Pre-Deep Learning State-of-the-Art	59
2. Application of Deep Learning to Four Vectors	61
3. Parameterized Networks	66
4. Handling Sets of Four Vectors	72
5. Physics-aware Networks	75
6. Conclusions	78
References	79

Chapter 4. Anomaly Detection for Physics Analysis and Less Than Supervised Learning 85

Benjamin Nachman

1. Introduction	85
2. Model Dependence in HEP Data Analysis	86
3. Signal Independent, Background Model Dependent	91
4. Supervised Approaches	92
5. Unsupervised Approaches	93
6. Weak Supervision and Topic Modeling	95
7. Hybrid Approaches	100
8. Results with Collider Data	102
9. Conclusions and Outlook	104
References	106

Part II: Data Quality Monitoring 113

Chapter 5. Data Quality Monitoring Anomaly Detection 115

*Adrian Alan Pol, Gianluca Cerminara,
Cecile Germain and Maurizio Pierini*

1. Introduction 116
2. Data Quality Monitoring for the LHC Experiments 117
3. Machine Learning Anomaly Detection for HEP DQM . . . 124
4. Detector Components Anomaly Detection with
Convolutional Neural Networks and Autoencoders 127
5. Data Certification Novelty Detection with Deep
Autoencoders 134
6. Trigger Rate Anomaly Detection with Conditional
Variational Autoencoders 136
7. LHC Monitoring with LSTMs 143
8. Conclusion 146
- References 146

Part III: Generative Models 151

Chapter 6. Generative Models for Fast Simulation 153

*Michela Paganini, Luke de Oliveira,
Benjamin Nachman, Denis Derkach,
Fedor Ratnikov, Andrey Ustyuzhanin
and Aishik Ghosh*

1. Generative Models for the Simulation of Particle Showering
in Calorimeters 153
2. Conclusions and Outlook 179
- References 180

Chapter 7. Generative Networks for LHC Events 191

Anja Butter and Tilman Plehn

1. Introduction	191
2. Generative Networks	196
3. Neural Networks in Event Generators	204
4. GANs and VAEs as Event Generators	215
5. Inverting the Simulation Chain	227
6. Outlook	234
References	236

Part IV: Machine Learning Platforms 241

Chapter 8. Distributed Training and Optimization of Neural Networks 243

Jean-Roch Vlimant and Junqi Yin

1. Introduction	243
2. Neural Network Optimization Formalism	245
3. Parameter Distribution	248
4. Data Distributed Training	250
5. Model Parallelism	253
6. Hyperparameter Optimization	255
7. Summary and Discussion	258
References	261

Chapter 9. Machine Learning for Triggering and Data Acquisition 265

Philip Harris and Nhan Tran

1. Introduction	265
2. Fast ML for Real-Time Readout and Near-Detector Triggering	271
3. Fast ML for Data Processing	280
4. Concluding Remarks	300
References	301

Part V: Detector Data Reconstruction **311**

Chapter 10. End-to-End Analyses Using Image Classification **313**

Adam Aurisano and Leigh H. Whitehead

1. Introduction	313
2. Traditional Workflow	314
3. Deep Learning Approaches	315
4. Convolutional Neural Networks in Lattice-Structured Experiments	324
5. Convolutional Neural Networks in Heterogeneous Collider Detectors	335
6. End-to-End Analysis of Time Series Using One-Dimensional CNNs	338
7. Graph Neural Networks for Large Three-Dimensional Detectors	340
8. Opening the Black-Box	341
9. Conclusions	349
References	350

Chapter 11. Clustering **355**

Kazuhiro Terao

1. Introduction	355
2. Fixed Number of Partitions	359
3. Convolutional Neural Networks for Pixel Clustering	368
4. Clustering Particles Using Graph Neural Networks	375
5. Summary	381
References	382

Chapter 12. Graph Neural Networks for Particle Tracking and Reconstruction **387**

Javier Duarte and Jean-Roch Vlimant

1. Introduction	387
2. Point Cloud and Graph Data	391

3. Graph Neural Networks	396
4. GNN Design Considerations	402
5. Applications to Particle Physics Tasks	406
6. Summary	423
References	425

Part VI: Jet Classification and Particle Identification from Low Level 437

Chapter 13. Image-Based Jet Analysis 439

Michael Kagan

1. Introduction	439
2. Jets and Jet Physics Challenges	442
3. Jet Images and Preprocessing	446
4. Computer Vision and Convolutional Neural Networks	449
5. Jet Tagging	454
6. Understanding Jet Image-Based Tagging	480
7. Other Applications of Jet Images	485
8. Conclusion	491
References	492

Chapter 14. Particle Identification in Neutrino Detectors 497

Ralitsa Sharankova and Taritree Wongjirad

1. Introduction	497
2. Behavior of Particles in Matter	498
3. Neutrino Interactions with Matter	501
4. Scintillator Detectors	502
5. Cherenkov Ring Imaging Detectors	513
6. Tracking Detectors	519
7. Concluding Thoughts	536
References	538

Chapter 15. Sequence-Based Learning 541

Rafael Teixeira de Lima

1. Introduction	541
2. Applications of RNNs to Jet Physics	546
3. Alternatives to RNNs	565
4. Conclusion	572
References	573

Part VII: Physics Inference 577

Chapter 16. Simulation-Based Inference

Methods for Particle Physics

579

Johann Brehmer and Kyle Cranmer

1. Particle Physics Measurements as a Simulation-Based Inference Problem	579
2. Inference with Surrogates	586
3. Inference with Sufficient Summary Statistics	594
4. Diagnostics, Calibration, and Systematic Uncertainties	597
5. Probabilistic Programming	599
6. Software and Computing	602
7. Summary	604
References	605

Chapter 17. Dealing with Nuisance Parameters 613

T. Dorigo and P. de Castro Manzano

1. Introduction	613
2. Nuisance-Parameterized Models	626
3. Feature Decorrelation, Penalized Methods, and Adversary Losses	629
4. Semi-supervised Approaches	638
5. Inference-Aware Approaches	642
6. Outlook	657
References	657

Chapter 18. Bayesian Neural Networks 663

*Tom Charnock, Laurence Perreault-Levasseur
and François Lanusse*

1. Introduction	664
2. Bayesian Neural Networks	667
3. Practical Implementations	680
4. Concluding Remarks and Outlook	710
References	711

Chapter 19. Parton Distribution Functions 715

Stefano Forte and Stefano Carrazza

1. Introduction	715
2. The State of the Art	724
3. The Future of PDFs in a Deep Learning Framework	737
References	759

Part VIII: Scientific Competitions and Open Datasets 763

Chapter 20. Machine Learning Scientific Competitions and Datasets 765

David Rousseau and Andrey Ustyuzhanin

1. Introduction	765
2. HiggsML	767
3. Flavor of Physics	772
4. TrackML	782
5. LHC Olympics	793
6. Competitions Platforms	797
7. Open Datasets and Repositories	805
8. Guidelines for New Competition Organizers	806
9. Conclusion	808
References	809

Index 813

Chapter 1

Introduction

Paolo Calafiura^{*,§}, David Rousseau^{†,¶} and Kazuhiro Terao^{‡,||}

**Lawrence Berkeley National Laboratory
1 Cyclotron Rd, Berkeley, California 94720, USA*

*†Université Paris-Saclay, CNRS/IN2P3, IJCLab
91405 Orsay, France*

*‡SLAC National Accelerator Laboratory
2575 Sand Hill Rd., Menlo Park, California 94025, USA*

§pcalafiura@lbl.gov

¶david.rousseau@ijclab.in2p3.fr

||kterao@slac.stanford.edu

1. Artificial Intelligence at the Frontiers of High-Energy Physics

Traditional Artificial Intelligence and Machine Learning (AI/ML) approaches have been applied to High-Energy Physics for decades [1–3], and they played a role in the Higgs boson discovery in 2012. In recent years, following the growth in these approaches in industry, modern AI/ML methods, such as deep learning, have become the state-of-the-art in several areas of Computational High-Energy Physics (HEP). For example, Boosted Decision Trees (Chapter 2), which are an integral part of many HEP analyses, are increasingly replaced by deep neural networks operating on high-level physics objects (Chapter 3).

Accelerated AI/ML platforms for training and inference (Part IV) have been crucial for recent advancements in AI for HEP. For example, neutrino experiments have shown how Convolutional Neural Networks outperform traditional algorithms on image-like data (Chapter 10). Deep learning methods are also used for Particle Identification and Jet Classification (Part VI). The opportunity to train large-scale generative models (Part III) may allow replacing traditional Monte Carlo simulation in some of the most resource-intensive HEP applications.

As the field develops, so does the potential for more transformative changes, including graph networks that capture the structure of experimental data (Chapters 11 and 12), and anomaly detection methods that monitor its quality (Chapter 5). New data analysis paradigms such as model-independent resonance searches (Chapter 4) and likelihood-free inference (Chapter 16) promise to increase the physics reach of HEP experiments. As analysis methods become more sophisticated, principled strategies to quantify and minimize statistical and systematic uncertainties associated with their predictions (Chapters 17 and 18) will be key. AI/ML methods also have a place in theoretical problems, such as estimating parton distribution functions, which cannot be computed from first principles QCD alone and need to be determined using experimental data (Chapter 19).

Finally, the organization of open AI/ML challenges and the availability of associated datasets (Chapter 20) have helped create a research community of physicists, data scientists, and computer scientists, several of whom have contributed to this book.

2. Why This Book, and Who Should Read It?

This book aims to close the gap between generic AI/ML textbooks and research papers discussing AI/ML applications in HEP. Can deep neural networks be used to reconstruct data and infer physics better than a traditional approach? What can we find using anomaly detection techniques on data from particle colliders? How are generative models used to support physics modeling? Can Bayesian

neural networks help infer uncertainties on physics measurement? These are a few of many research questions faced and addressed by physicists with AI/ML expertise. Through topical reviews written by leading experts, this book guides HEP researchers to apply state-of-the-art methods to their challenges and hopefully provide them with the foundations to push the boundaries and develop new AI/ML techniques.

Target readers include students, postdocs, and experienced physicists looking for an overview of AI/ML techniques to apply to their research. The book may also be useful to data scientists, ML experts, and scientists from other domains interested in how Computational HEP has embraced modern AI/ML methods.

3. Pre-requisites

This book assumes a basic working knowledge of AI/ML (including linear models, gradient-based optimization and regularization, and neural networks), data analysis (e.g. function fitting, likelihood maximization, and error analysis), and introductory particle physics. Readers who have contributed to an HEP analysis using classic statistical tools and are familiar with basic ML concepts should benefit most from this book.

Many online courses provide an excellent introduction to modern AI/ML, including [4]. Several textbooks [5–7] complement them by covering in detail AI/ML conceptual foundations and applications.

Finally, the HEP community organizes several schools and workshops (e.g. [8]) that provide lectures and tutorials on basic and advanced “AI for HEP” topics.

4. How to Use This Book

This book is organized by physics application. Using the table of contents and the abstract at the beginning of each chapter, the reader can navigate directly to the part or chapter relevant to their application. Each chapter is self-contained and provides an introduction to the concepts, practical examples of their applications to HEP,

and a survey of recent research. Given how fast the field is evolving, some emerging topics have not been included,^a however, authors have striven to cover the most recent public studies.

As the book covers the many steps required to derive a physics result from the analysis of HEP data, it could also be used as a textbook for a course about AI/ML applications to HEP data analysis.

Furthermore, this book surveys many research challenges across various sub-fields in particle physics, including cosmology, neutrino physics, collider physics, and particle phenomenology. We hope that it will encourage the reader to join our vibrant research community.

Acknowledgments

The editors are grateful to the authors for the quality of their contributions and the collaborative spirit with which they helped improve and harmonize the whole book. In particular, we would like to thank Anja Butter, Tom Charnock, Yann Coadou, Pablo de Castro, Tommaso Dorigo, Javier Duarte, Stefano Forte, Cécile Germain, Aishik Ghosh, Phil Harris, Michael Kagan, Ben Nachman, Michela Paganini, Tilman Plehn, Adrian Pol, Peter Sadowski, Jean-Roch Vlimant, and Leigh Whitehead for reviewing one or more chapters. A special thanks to our guest reviewers Wahid Bhimji and Roman Kogler. Finally, we are grateful to Elena Nash and Kah-Fee Ng at World Scientific for their patience with an inexperienced editorial team and all their support.

References

- [1] B. H. Denby, Neural networks and cellular automata in experimental high-energy physics, *Comput. Phys. Commun.* **49** (1988) 429–448; doi: 10.1016/0010-4655(88)90004-5.
- [2] C. Peterson, Track finding with neural networks, *Nucl. Instrum. Methods Phys. Res. A* **279** (1989) 537; doi: 10.1016/0168-9002(89)91300-4.
- [3] G. Stimpff-Abele and L. Garrido, Fast track finding with neural networks, *Comput. Phys. Commun.* **64** (1991) 46–56; doi: 10.1016/0010-4655(91)90048-P.

^aFor that, the reader is referred to the IML working group living review [9].

- [4] A. Ng, Machine learning by Stanford. URL <https://www.coursera.org/learn/machine-learning>.
- [5] C. M. Bishop, *Pattern Recognition and Machine Learning*, Information Science and Statistics (Springer, New York, 2006). URL <https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/>.
- [6] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning* (MIT Press, 2016). URL <http://www.deeplearningbook.org>.
- [7] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics (Springer, New York, 2001). URL <https://web.stanford.edu/~hastie/ElemStatLearn/>.
- [8] D. Rousseau, G. M. Innocenti, L. Moneta, P. Vischia, S. Akar, R. Torre and A. Wulzer, Inter-experimental LHC machine learning working group. URL <https://iml.web.cern.ch/>.
- [9] M. Feickert and B. Nachman (eds.), A living review of machine learning for particle physics. URL <https://iml-wg.github.io/HEPML-LivingReview/>.

This page intentionally left blank

Part I

Discriminative Models for Signal/Background Boosting

This page intentionally left blank

Chapter 2

Boosted Decision Trees

Yann Coadou

*Centre de physique des particules de Marseille (CPPM),
Aix Marseille Université, CNRS/IN2P3, Marseille, France
coadou@cppm.in2p3.fr*

Boosted decision trees are a very powerful machine learning technique. After introducing specific concepts of machine learning in the high-energy physics context and describing ways to quantify the performance and training quality of classifiers, decision trees are described. Some of their shortcomings are then mitigated with ensemble learning, using boosting algorithms, in particular AdaBoost and gradient boosting. Examples from high-energy physics and software used are also presented.

1. Introduction

Decision trees are a machine learning technique that appeared in the mid-1980s and are still the subject of advanced studies in the field. Because it is a sophisticated supervised multivariate technique, learning from examples, it is important to remember that before applying it to real data (e.g. collisions from a high-energy physics experiment), it is crucial to have a good understanding of the data and of the physics model used to describe them (simulated samples, reconstruction and identification efficiencies, etc.). Any discrepancy between the real data and physics model (that is, features in the data that are not reproduced by the physics model because the simulation is incorrect or because the real data were not properly groomed) will provide an artificial separation that the decision trees will use, misleading the analyzer. The hard (and interesting) part of the analysis is in building the proper physics model, not in “just” extracting the signal. But once this is properly done, decision trees (and especially

their boosted versions) provide a very powerful tool to increase the significance of any analysis.

Ever since their first use by the MiniBooNe collaboration for analysis and particle identification [1, 2] and by the D0 experiment for the first evidence of single top quark production [3, 4], boosted decision trees have been a primary tool in high-energy physics to increase the discovery potential and measurement precision of experiments, in particular at the Tevatron and at the LHC. They are still highly relevant (and highly performing) in 2021, even though deep neural networks are becoming a serious contender.

As this is the first chapter of this book, some of the basic concepts useful in the context of high-energy physics when using most techniques presented in this and other chapters are summarized in Sec. 2. Section 3 explains how a decision tree is constructed, what parameters can influence its development and what its intrinsic limitations are. One possible extension of decision trees, boosting, is described in detail in Sec. 4, and other techniques trying to reach the same goal as boosting are presented in Sec. 5. Popular software implementations are introduced in Sec. 6, before reaching conclusions in Sec. 7.

2. Specificity of High-Energy Physics

All techniques presented in this book need to learn from examples. After a short list of definitions to have a common language between the physicist and the computer scientist in Sec. 2.1, several training strategies are presented in Sec. 2.2, as well as how to deal with the samples to minimize training bias and maximize statistical power. In order to properly assess the performance of a classifier, cross-validation is introduced in Sec. 2.3. Section 2.4 describes typical usage of machine learning algorithms in high-energy physics. Several figures of merit are described in Sec. 2.5 and overtraining is addressed in Sec. 2.6.

2.1. Terminology

Here are a few terms that take on different meanings in a high-energy physics or machine learning context.

Event All information collected during a collision inside a detector, or reproduced from a Monte Carlo simulation of such collisions (equivalent to a “sample” in machine learning literature).

Sample A collection of events, a dataset.

Variable A property of the event or of one of its constituents (“feature” in machine learning).

Cut To cut on a variable is to apply a threshold on this variable and keep only events satisfying this condition. A cut-based analysis is applying such thresholds on several variables to select events.

Event weight In high-energy physics events usually have an associated weight, which depends on how many events were generated (relating to the process cross section and collected luminosity) and various corrections applied to simulations to account for differences between data and Monte Carlo predictions (jet energy scale or object identification efficiency are such weights). When using machine learning techniques all events are often treated equal by default. It is therefore important for the physicist to make sure to give the proper initial weight to all its input events. Then machine learning algorithms may internally reweight the events for their own purpose, but the starting point will correspond to the physical distributions. The concept is similar to importance weighting in machine learning, where events are given a larger weight to account, for instance, for their scarcity in the training sample.

2.2. *Splitting samples for training*

Decision trees, as many of the techniques presented in this book, belong to the class of algorithms using supervised learning: during training, the classifier is presented only with events for which it knows features (discriminating variables) and class label (for instance in the binary case, whether the event is signal or background).

In order to not introduce bias, it is important to use an independent set of events during training, events that are then not used

when performing a measurement. The usual approach is to split the dataset in three parts: a training sample from which to learn the task, a validation sample to evaluate performance and possibly optimize the classifier hyperparameters, and a testing sample for the actual measurement. In high-energy physics, simulated Monte Carlo events are often used for these three samples, and the performance on the testing sample is compared to that on data collected from the detector (never seen during training). Discrepancies between testing sample and data introduce a potential pitfall, that can be addressed with transfer learning and domain adaptation [5].

In general labeled data are “expensive” to produce: hiring people to label images or translate speech, collecting X-ray images and medical diagnosis, etc. In high-energy physics very accurate, though not perfect, event generators and detector simulators are available. Models can be trained on the samples they provide, which are however quite costly in resources so that they should be used with parsimony. At the same time an increasing training set size is often associated with improved classifier performance. Monte Carlo samples can be split in half, one half for training (holding out part of this dataset for validation) and one for testing. By doing this half of the sample is “wasted”, not used for either training or testing, decreasing the quality of the training and of the measurement. The use of the sample can be maximized by performing two trainings: train the same classifier on the two halves (say, one on events with an even event number and one on events with an odd event number), and when testing, apply the classifier which did not see the event during training (so, the one trained on odd events is applied on even ones, and vice versa). The concept can be generalized to any number of splits, increasing the number of trained classifiers, each of them using a larger fraction of the available dataset for training.

2.3. *Cross-validation*

Training machine learning algorithms is usually a stochastic problem, the randomness coming from the training sample content, the optimization process or the technique itself. This means that when

training only once, there is a possibility to obtain an “abnormal” result by chance, too good or too bad compared to what could be expected. In high-energy physics some training samples may be limited in size and become very sensitive to this issue. To get a proper estimate of the mean performance and associated uncertainty (the variability of the algorithm output, originating from the training procedure), it may be better to perform several trainings. This principle was introduced with the so-called K -fold cross-validation, originally for decision trees [6]. After dividing a training sample \mathcal{L} into K subsets of equal size, $\mathcal{L} = \bigcup_{k=1,\dots,K} \mathcal{L}_k$, a classifier T_k is trained on the $\mathcal{L} - \mathcal{L}_k$ sample and tested on \mathcal{L}_k . This produces K classifiers, from which the mean performance and associated uncertainty is extracted. It helps in choosing the best model (each being tested with cross-validation), rather than relying on a single training for each model (which may or may not have an upward or downward performance fluctuation). Once the model is chosen, it can be retrained on the full (larger) training set, assuming its performance should approach the observed mean performance.

2.4. Using machine learning

This book describes various ways of using machine learning in high-energy physics to accomplish many different tasks. Boosted decision trees are mostly used to separate a rare signal from a large background in physics analyses or to identify physics objects in the detector (see several use cases in Sec. 4.8.1). In practice, these results are obtained in two ways. By applying a threshold on the boosted decision tree output a region or working point can be defined, as shown in Fig. 1(a): cutting at 0.83 defines a b -tagging working point with 70% efficiency on b -jets and a rejection factor (defined as the inverse of efficiency) of 313 (8) against light-flavored jets (c -jets) [7]. The second approach consists in using the shape of the boosted decision tree output as the discriminating variable for the final analysis. As an example, in Fig. 1(b) the bins of “BDT score” for all components of the physics model are included in a binned likelihood fit to the data. The low score values help constrain the background, and the

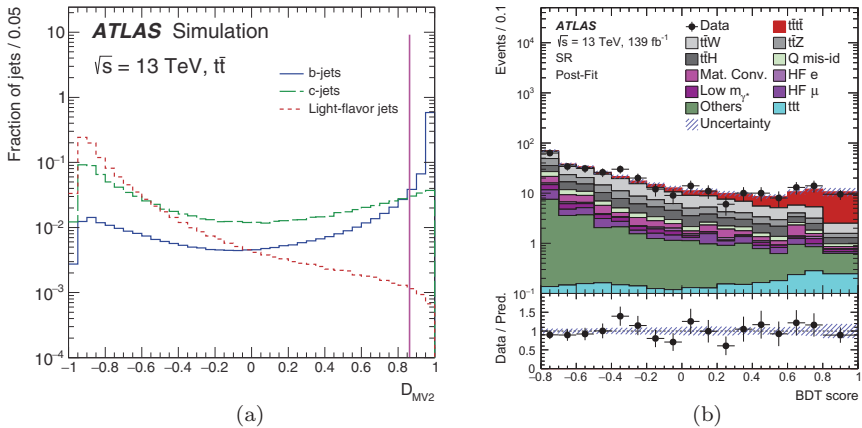


Fig. 1. (a) Output of the boosted decision tree used to identify jets originating from b -quarks in ATLAS [7]. (b) Boosted decision tree output used in a fit between data and physics model to extract the $t\bar{t}t\bar{t}$ signal [8].

high score bins reveal the need of the signal contribution to match the data, leading to the first evidence for the production of $t\bar{t}t\bar{t}$ in ATLAS [8].

2.5. Figures of merit

It is nowadays very easy, in just a few lines, to write the code to train and apply various machine learning algorithms, with several software options on the market (see Sec. 6). The lengthy part is more in the design and optimization of the model itself (that is, what algorithm, structure, hyperparameters to put in these few lines), and how to pick the best one. Several measures that are commonly used, in particular in high-energy physics, are presented below.

2.5.1. ROC curve and area under the curve

The receiver operating characteristic curve, or ROC curve, is a representation of the capacity of a binary classifier to separate the two classes, as its discrimination threshold is varied. It is plotting the true positive rate (or recall, a measure of the proportion of actual positives that are correctly identified as such) against the false positive

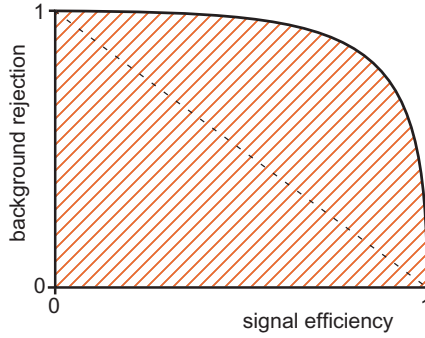


Fig. 2. Example ROC curve. The hatched area is the area under the curve. The dashed line corresponds to random guessing.

rate (or fall-out, actual negatives improperly identified as positive), obtained when scanning the classifier output. In the context of signal and background, it shows signal efficiency vs. background efficiency (or background rejection, defined as $1 - \text{efficiency}$). An example is shown in Fig. 2. In this convention, the better the classifier, the closer the curve is to the top right corner. The dashed line in the middle represents the performance of a classifier that is randomly guessing, rejecting or accepting 50% of signal and background in all cases. This is the worst achievable performance.

To compare ROC curves between classifiers, the area under the curve, AUC (hatched area in Fig. 2) can be computed. Perfect separation gives an AUC of one, while random guessing corresponds to an AUC of 0.5.

A single number summary is of course practical, but hides details of the ROC curves being compared. If one ROC curve is systematically above the other, its AUC is larger and reflects better performance across the board. But if two ROC curves cross each other, then the interpretation of the AUC is more tricky: depending on the usage of the classifier, a higher curve at high background rejection may be more interesting than one at high efficiency for instance, so how to interpret the AUC is up to the analyzer. To partially account for this effect, it is also possible to compute the AUC only above a certain threshold.

2.5.2. Significance

In a physics analysis, the AUC is rarely the number of interest to optimize. It is more typical to aim for the best cross-section significance $\frac{s}{\sqrt{s+b}}$ or excess significance $\frac{s}{\sqrt{b}}$, where s (b) is the sum of weights (see Sec. 2.1) of signal (background) events. With n events in data, the observed significance is obtained by replacing s by $n - b$. Given a machine learning algorithm output, typically in the range $[0, 1]$ or $[-1, 1]$, as is done when producing the ROC curve, s and b are computed above a threshold on the discriminant output, scanning its full range. It usually goes through a maximum towards high-output values, before decreasing when statistics become too small. This maximum significance corresponds to the optimal value on which to cut on the discriminant to get the best possible analysis.

This simple-minded formula is very popular in high-energy physics but has shortcomings, and a refined version (counting experiment supposing a single Poisson distributed value, with known background) gives the approximate median significance [9]:

$$\text{AMS} = \sqrt{2 \left((s + b) \ln \left(1 + \frac{s}{b} \right) - s \right)}.$$

Expanding the logarithm in s/b leads back to the previous formula, qualifying the validity of the approximation (requires $s \ll b$):

$$\text{AMS} = \frac{s}{\sqrt{b}} (1 + \mathcal{O}(s/b)).$$

Optimizing the AMS corresponds to optimizing the ROC curve, focusing on the region with very high background rejection. This is the typical regime of a physics analysis.

There is usually an uncertainty on the background, which affects the significance. To extract their final results, modern analyses rely on advanced statistical models with a complex machinery (usually based on the RooStat framework [10]) accounting for all possible systematic effects. Running this whole infrastructure during machine learning training optimization is usually prohibitive (complexity, CPU cost), so a simpler proxy to the analysis performance measure is necessary.

The simplest way to account partially for background uncertainty ($\sigma_b \equiv ||b - b_{\text{syst}}||$) is to replace \sqrt{b} by the quadratic sum of \sqrt{b} and σ_b :

$$\frac{s}{\sqrt{b + \sigma_b^2}}.$$

A refined version of the AMS can also take into account the background uncertainty [11]:

$$\text{AMS}_1 = \sqrt{2 \left((s + b) \ln \frac{s + b}{b_0} - s - b + b_0 \right) + \frac{(b - b_0)^2}{\sigma_b^2}},$$

with $b_0 = \frac{1}{2} \left(b - \sigma_b^2 + \sqrt{(b - \sigma_b^2)^2 + 4(s + b)\sigma_b^2} \right).$

Expanding in powers of s/b and σ_b^2/b gives back the simpler formula:

$$\frac{s}{\sqrt{b + \sigma_b^2}} (1 + \mathcal{O}(s/b) + \mathcal{O}(\sigma_b^2/b)).$$

Finally, to account for the shape of the discriminant rather than only choosing the best cut in a counting experiment, it is possible to replace the global counts s , b and σ_b by their counts in each bin, summing up contributions of N bins of discriminant output:

$$\text{AMS}_1^{\text{sum}} = \sqrt{\sum_i^N \left(2 \left((s_i + b_i) \ln \frac{s_i + b_i}{b_{0i}} - s_i - b_i + b_{0i} \right) + \frac{(b_i - b_{0i})^2}{\sigma_{bi}^2} \right)},$$

$b_{0i} = \frac{1}{2} \left(b_i - \sigma_{bi}^2 + \sqrt{(b_i - \sigma_{bi}^2)^2 + 4(s_i + b_i)\sigma_{bi}^2} \right).$

2.6. Controlling overtraining

Overtraining is what happens when a classifier learns too much about the specific details of the training sample, while these features are not representative of the underlying distributions. It may then be targeting noise, or misrepresent regions with too little statistics to train on. When applying such a classifier on the testing sample, its performance will be worse than that of a classifier immune to this issue,

because it does not generalize well. It should be noted that what is often called overtraining here and in the following, in accordance with high-energy physics usage, is usually referred to as overfitting in the machine learning community. This is the so-called bias–variance trade-off [12]: it is difficult to minimize both the bias (the difference between the prediction of the model and the correct value it tries to predict) and variance (the variability of the model prediction for a given event, when considering multiple realizations of this same model). Increasing model complexity lowers the bias while increasing variance.

A particular type of overtraining is very easy to avoid, by following good practices from Sec. 2.2: never use training events when making the final measurement, which has to be performed on an independent set of events, never seen during training. Otherwise, the performance will be artificially enhanced on the “testing” sample and comparisons with the application to data will be impossible (or worse if not noticed).

Several techniques exist to mitigate overtraining, generically referred to as regularization. They typically add a penalty for complexity to the loss function that is minimized during training (the function that maps each event to a real number quantifying the difference between the predicted and true classes or values). With classifier f and loss function L , a regularization term $R(f)$ is added to the loss function, which becomes $L(f) + \lambda R(f)$, where λ is a parameter controlling the importance of the regularization term. This will favor simpler models (increasingly simpler with larger values of λ , with the risk of underfitting with too much regularization), less susceptible to overtraining. $R(f)$ can take various forms, like L1 (L2) regularization based on the sum of weights (sum of squared weights) used to describe neural networks, or the number and depth of trees (see Sec. 3.5.2). Sparsity (setting many weights to zero [13]) and dropout (randomly dropping out nodes during training [14]) are more recent very effective approaches for neural networks. Ensemble learning (see Secs. 3.5.3 and 5) is another approach.

It is important to check whether the model suffers from overtraining. As shown in Fig. 3 this can be achieved by monitoring the error

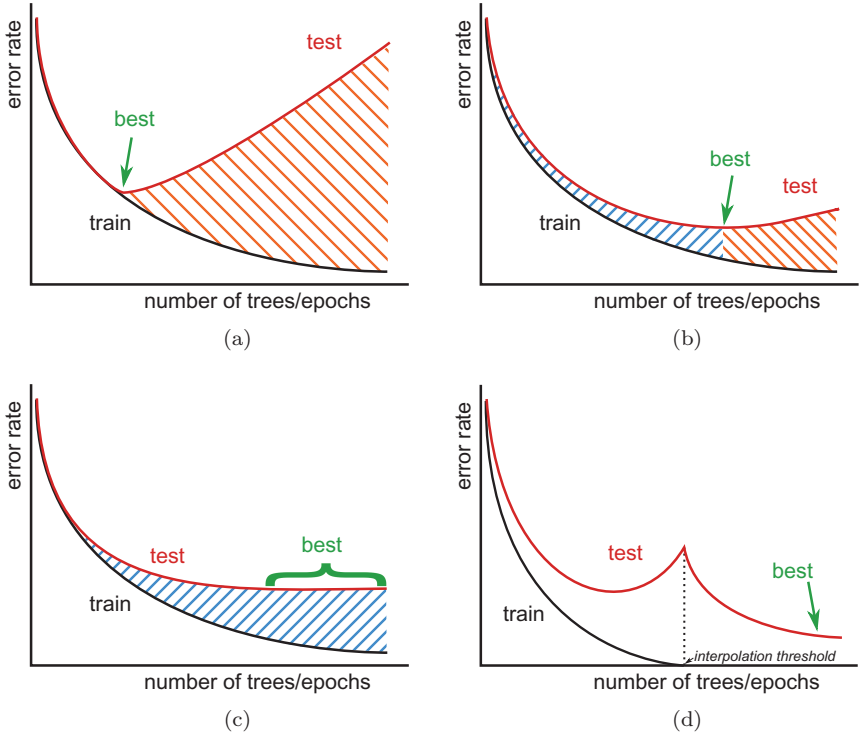


Fig. 3. Overtraining estimation using the error rate as a function of the number of trees (for boosted decision trees) or epochs (for neural networks). Black curves are measured on the training sample and red curves on the validation sample. The optimal classifier corresponds to the “best” label. The hatched areas represent overtraining: beneficial in blue (but underfitting), detrimental in orange (overfitting). (a) Typical curves, with the best model at the minimum of the testing curve, and overfitting beyond with decrease of performance. (b) The best model is overtrained but still improves performance. (c) Typical curves for boosted decision trees with flattening testing error rate: all models in the flat area perform equally well despite increasing overtraining. (d) Interpolation regime: the best classifier is obtained after the training error has reached zero.

rate (or the loss function) during training, as a function of the number of trees with boosted decision trees or training epoch with neural networks, on the training and validation samples. Figure 3(a) is the canonical example of such curves. The training error tends towards zero, while the testing curve first follows the training curve, reaches a minimum and increases again. The best classifier is the one at the

minimum, training further will reduce performance and cause overfitting: the classifier has too much capacity (complexity) with respect to the training sample. Selecting the model at the minimum means early stopping [12].

In many cases though, the situation is similar to Fig. 3(b): the training and testing curves follow each other but start diverging while still both improving. The classifier is therefore already learning specificities of the training set, but still learning properties that generalize well and improve performance on the validation set. The testing curve goes through a minimum, corresponding to the best model, and increases again, this time showing detrimental overtraining as the performance decreases on the validation set (overfitting regime). This is the typical U-shaped curve arising from the bias-variance trade-off.

The curves could also look like Fig. 3(c), where the testing curve never goes through a minimum and instead flattens. Once in the plateau, all classifiers are equivalent in terms of performance on the validation set, while the training error keeps improving (and could reach zero, this is the so-called interpolation regime [15]). This is a typical curve for boosted decision trees.

Finally the situation could correspond to Fig. 3(d). At the interpolation threshold the training error reaches zero, but continued training of high-capacity classifiers leads to a double descent curve: the testing performance keeps increasing while the training error stays at zero [16].

3. Decision Trees

Decision trees are a machine learning technique first developed in the context of data mining and pattern recognition [6], which then gained momentum in various fields, including medical diagnosis [17, 18], insurance and loan screening, or optical character recognition of handwritten text [6].

It was developed and formalized by Breiman *et al.* [6] who proposed the CART algorithm (Classification And Regression Trees) with a complete and functional implementation of decision trees.

The basic principle is rather simple: it consists in extending a simple cut-based analysis into a multivariate technique by continuing to analyze events that fail a particular criterion. Many, if not most, events do not have all characteristics of either signal or background (for a two-class problem). The concept of a decision tree is therefore to not reject right away events that fail a criterion, and instead to check whether other criteria may help to classify these events properly.

In principle, a decision tree can deal with multiple output classes, each branch splitting in many subbranches. In this chapter, almost only binary trees will be considered, with only two possible classes: signal and background. The same concepts generalize to non-binary trees, possibly with multiple outputs.

Section 3.1 describes the decision tree building algorithm, controlled by hyperparameters presented in Sec. 3.2. The way to split nodes is explained in Sec. 3.3, while Sec. 3.4 describes how decision trees can advantageously deal with input variables and how to optimize their list. Finally Sec. 3.5 reports several shortcomings of decision trees, with suggestions to address them.

3.1. *Algorithm*

Mathematically, decision trees are rooted binary trees (as only trees with two classes, signal and background, are considered). An example is shown in Fig. 4. A decision tree starts from an initial node, the root node. Each node can be recursively split into two daughters or branches, until some stopping condition is reached. The different aspects of the process leading to a full tree, indifferently referred to as growing, training, building or learning, are described in the following sections.

Consider a sample of signal (s_i) and background (b_j) events, each with weights w_i^s and w_j^b , respectively, described by a set \vec{x}_i of variables. This sample constitutes the root node of a new decision tree.

Starting from this root node, the algorithm proceeds as follows:

- (1) If the node satisfies any stopping criterion, declare it as terminal (that is, a leaf) and exit the algorithm.

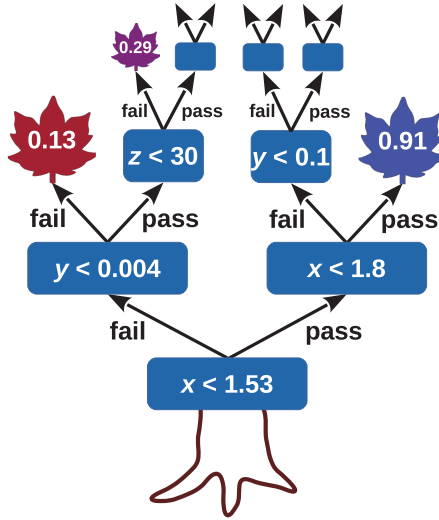


Fig. 4. Graphical representation of a decision tree. Blue rectangles are internal nodes with their associated splitting criterion; leaves are terminal nodes with their purity.

- (2) Sort all events according to each variable in \vec{x} .
- (3) For each variable, find the splitting value that gives the best separation between two children, one with mostly signal events, the other with mostly background events (see Sec. 3.3 for details). If the separation cannot be improved by any splitting, turn the node into a leaf and exit the algorithm.
- (4) Select the variable and splitting value leading to the best separation and split the node in two new nodes (branches), one containing events that fail the criterion and one with events that satisfy it.
- (5) Apply recursively from step 1 on each node.

This is a greedy algorithm, not guaranteed to find the optimal solution. At each node, all variables can be considered, even if they have been used in a previous iteration: this allows to find intervals of interest in a particular variable, instead of limiting oneself to using each variable only once.

It should be noted that a decision tree is human readable: exactly which criteria an event satisfied in order to reach a particular leaf can be traced. It is therefore possible to interpret a tree in terms of, e.g., physics, defining selection rules, rather than only as a mathematical object.

In order to make the whole procedure clearer, let us take the tree in Fig. 4 as an example. Consider that all events are described by three variables: x , y and z . All signal and background events make up the root node.

All events are first sorted according to each variable:

$$\begin{aligned} x^{s_1} &\leq x^{b_{34}} \leq \dots \leq x^{b_2} \leq x^{s_{12}}, \\ y^{b_5} &\leq y^{b_3} \leq \dots \leq y^{s_{67}} \leq y^{s_{43}}, \\ z^{b_6} &\leq z^{s_8} \leq \dots \leq z^{s_{12}} \leq z^{b_9}, \end{aligned}$$

where superscript s_i (b_j) represents signal (background) event i (j). Using some measure of separation between classes (see below) the best splitting for each variable may be (arbitrary unit):

$$\begin{aligned} x &< 1.53 \quad \text{separation} = 5, \\ y &< 0.01 \quad \text{separation} = 3, \\ z &< 25 \quad \text{separation} = 0.7. \end{aligned}$$

The best split is $x < 1.53$, and two new nodes are created, the left one with events failing this criterion and the right one with events satisfying it. The same algorithm is applied recursively to each of these new nodes. As an example, consider the right-hand-side node with events that satisfied $x < 1.53$. After sorting again all events in this node according to each of the three variables, it was found that the best criterion was $x < 1.8$, and events were split accordingly into two new nodes. This time the right-hand-side node satisfied one of the stopping conditions and was turned into a leaf. From signal and background training events in this leaf, the purity was computed as $p = 0.91$. The left-hand-side node keeps splitting further.

The decision tree output for a particular event i is defined by how its \vec{x}_i variables behave in the tree:

- (1) Starting from the root node, apply the first criterion on \vec{x}_i .
- (2) Move to the passing or failing branch depending on the result of the test.
- (3) Apply the test associated to this node and move left or right in the tree depending on the result of the test.
- (4) Repeat step (3) until the event ends up in a leaf.
- (5) The decision tree output for event i is the value associated with this leaf.

There are several conventions used for the value attached to a leaf. It can be the purity $p = \frac{s}{s+b}$ where s (b) is the sum of weights of signal (background) events that ended up in this leaf during training. It is then bound to $[0, 1]$, close to 1 for signal and close to 0 for background.

It can also be a binary answer, signal or background (mathematically typically +1 for signal and 0 or -1 for background) depending on whether the purity is above or below a specified critical value (e.g. +1 if $p > \frac{1}{2}$ and -1 otherwise).

Looking again at the tree in Fig. 4, the leaf with purity $p = 0.91$ would give an output of 0.91, or +1 as signal if choosing a binary answer with a critical purity of 0.5.

3.2. *Tree hyperparameters*

The number of hyperparameters of a decision tree is relatively limited. The first one is not specific to decision trees and applies to most techniques requiring training: how to normalize signal and background with respect to each other before starting the training? Conventionally the sums of weights of signal and background events are chosen to be equal (balanced classes), giving the root node a purity of 0.5, that is, an equal mix of signal and background. Decision trees are not particularly sensitive to this original normalization as in practice, a few early splits will produce nodes with more balanced categories,

therefore only leading to a limited inefficiency in the training process which only impacts marginally the final discriminating power.

Other hyperparameters concern the selection of splits. A list of discriminating variables is needed, and a way to evaluate the best separation between signal and background events (the goodness of the split). Both aspects are described in more detail in Sec. 3.3 and Sec. 3.4.

The splitting has to stop at some point, declaring such nodes as terminal leaves. Conditions to satisfy can include:

- a minimum leaf size. A simple way is to require at least N_{\min} training events in each node after splitting, to ensure the statistical significance of the purity measurement, with a statistical uncertainty $\sqrt{N_{\min}}$. It becomes a little bit more complicated with weighted events, as is normally the case in high-energy physics applications. Using the effective number of events instead may be considered:

$$N_{\text{eff}} = \frac{\left(\sum_{i=1}^N w_i\right)^2}{\sum_{i=1}^N w_i^2},$$

for a node with N events associated to weights w_i ($N_{\text{eff}} = N$ for unweighted events).

- having reached perfect separation (all events in the node belong to the same class).
- an insufficient improvement with further splitting.
- a maximum tree depth, if the tree cannot have more than a certain number of layers (for purely computational reasons or to have like-size trees).

Finally a terminal leaf has to be assigned to a class. This is classically done by labeling the leaf as signal if $p > 0.5$ and background otherwise.

3.3. *Splitting a node*

The core of a decision tree algorithm resides in how a node is split into two. Consider an impurity measure $i(t)$ for node t , which describes

to what extent the node is a mix of signal and background. Desirable features of such a function are that it should be:

- maximal for an equal mix of signal and background (no separation).
- minimal for nodes with either only signal or only background events (perfect separation).
- symmetric in signal and background purities, as isolating background is as valuable as isolating signal.
- strictly concave in order to reward purer nodes. This tends to favor asymmetric end cuts with one smaller node and one larger node.

A figure of merit can be constructed with this impurity function, as the decrease of impurity for a split S of node t into two children t_P (pass) and t_F (fail):

$$\Delta i(S, t) = i(t) - p_P \cdot i(t_P) - p_F \cdot i(t_F),$$

where p_P (p_F) is the fraction of events that passed (failed) split S .

The goal is to find the split S^* that maximizes the decrease of impurity:

$$\Delta i(S^*, t) = \max_{S \in \{\text{splits}\}} \Delta i(S, t).$$

It will result in the smallest residual impurity, which minimizes the overall tree impurity.

A stopping condition can be defined using the decrease of impurity, not splitting a node if $\Delta i(S^*, t)$ is less than some predefined value. Such early-stopping criterion requires care, as sometimes a seemingly very weak split may allow child nodes to be powerfully split further (see Sec. 3.5.2 about pruning).

Common impurity functions (exhibiting most of the desired features mentioned previously) are illustrated in Fig. 5:

- the misclassification error: $1 - \max(p, 1 - p)$,
- the (cross) entropy [6]: $-\sum_{i=s,b} p_i \log p_i$, with $p_b = 1 - p_s$ and $p_s = p$,
- the Gini index of diversity [19].

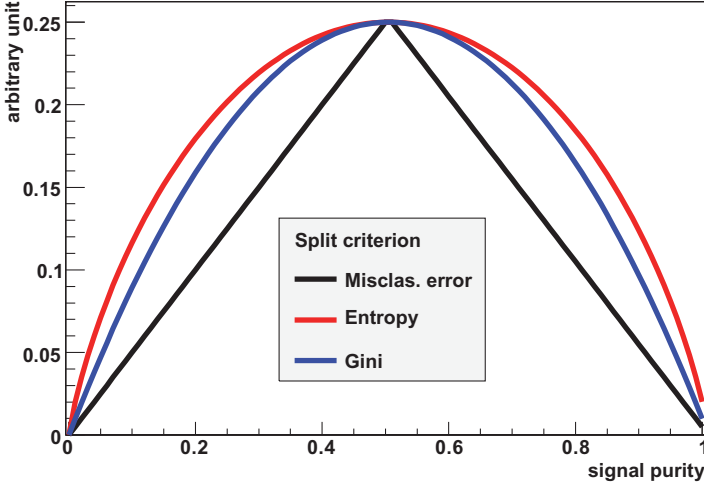


Fig. 5. Impurity measures as a function of signal purity.

The Gini index is the most popular in decision tree implementations. It typically leads to similar performance to entropy.

Other measures are also used sometimes, which do not satisfy all criteria listed previously but attempt at optimizing signal significance, a typical final goal in high-energy physics applications (see Sec. 2.5.2):

- cross-section significance (optimizing $\frac{s}{\sqrt{s+b}}$): $-\frac{s^2}{s+b}$,
- excess significance (optimizing $\frac{s}{\sqrt{b}}$): $-\frac{s^2}{b}$.

3.4. Variable selection

Overall decision trees are very resilient to most factors affecting variables. They are not too much affected by the “curse of dimensionality”, which forbids the use of too many variables in most multivariate techniques. For decision trees the CPU consumption scales as $nN \log N$ with n variables and N training events. It is not uncommon to encounter decision trees using tens [4] or hundreds [2] of variables, although this is usually frowned upon in high-energy physics: more variables means more distributions and correlations to check, more

complex interplay with systematic uncertainties, more dependence on the Monte Carlo event properties that are usually used during training and may not match real data so well, so physicists tend to reduce the list of discriminating variables to typically 10–15. On the other hand adding variables tends to always improve the performance of decision trees (see Sec. 4.8.1 for an example).

3.4.1. *Manipulating variables*

With most machine learning algorithms, a careful preparation of inputs is necessary to achieve good performance. Although not detrimental to decision trees, such manipulations are not really compulsory as decision trees tend to be very stable under such transforms.

A decision tree is immune to duplicate variables: the sorting of events according to each of them would be identical, leading to the exact same tree. The order in which variables are presented is completely irrelevant: all variables are treated equal. The order of events in the training samples is also irrelevant.

If variables are not very discriminating, they will simply be ignored and will not add any noise to the decision tree. The final performance will not be affected, it will only come with some CPU overhead during both training and evaluation.

Decision trees can deal easily with both continuous and discrete variables, simultaneously.

Another typical task before training a multivariate technique is to transform input variables by for instance making them fit in the same range (normalization), having unit variance (standardization) or taking the logarithm to regularize the variable. This is totally unnecessary with decision trees, which are completely insensitive to the replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them (e.g. converting MeV to GeV), as the same ordering of events would induce the same splits on the dataset, producing the same decision tree. This means that decision trees have some immunity against outliers. The above is strictly true only if testing all possible cut values while evaluating the optimal split. If there is some computational optimization (e.g., check only 20 possible cuts on each variable), it may not work

anymore and some transformation of inputs may be beneficial, at the very least to speed up convergence (numerical precision could also be a factor).

If linear correlations exist between variables, first decorrelating the input variables and then feeding them to the decision tree may help. If not doing this decorrelation, a decision tree will anyway find the correlations but in a very suboptimal way, by successive approximations, adding complexity to the tree structure without performance gain.

3.4.2. Mean decrease impurity

It is possible to rank variables in a decision tree, adding up the decrease of impurity (see Sec. 3.3) for each node where the variable was used to split, hence computing the mean decrease impurity (MDI). The variable with the largest decrease of impurity is the best variable. A shortcoming of this approach is that it is computed on the training set only, and may be exaggerating the importance of some variables because of overfitting.

There is another shortcoming with variable ranking in a decision tree: variable masking. Variable x_j may be just a little worse than variable x_i and would end up never being picked in the decision tree growing process. Variable x_j would then be ranked as irrelevant. But if x_i were removed, then x_j would become very relevant. Note that this is not important in terms of pure performance of the tree: it did find the optimal way to use both variables in this particular training. If trying to learn something from the tree structure on the other hand, like deriving selection rules, this phenomenon will interfere with the potential understanding.

There is a solution to this feature, called surrogate splits [6]. For each split, a comparison is made between training events that pass or fail the optimal split and events that pass or fail a split on another variable. The split that mimics best the optimal split is called the surrogate split. This can be taken into consideration when ranking variables. It has applications in case of missing data: the optimal split can be replaced by the surrogate split.

All in all, variable rankings should never be taken at face value. They do provide valuable information but should not be over-interpreted.

3.4.3. *Permutation importance*

The shortcomings of MDI discussed above are partially addressed with a different technique called permutation importance or mean decrease accuracy (MDA). While MDI mostly works for decision trees, permutation importance is suited for all models using tabular data. It is defined as the decrease of performance of an already trained model when applying it on a sample after randomly shuffling a single discriminating variable [20]. If the variable is of any use, the performance should decrease when submitted to this noisy input, and more so if the tree relies heavily on this feature for its prediction. Repeating this for all input variables, the importance of each of them can be ranked. The operation can be done multiple times, shuffling each variable differently, in order to get a mean value and uncertainty on variable importance. As with MDI, the measured importance is not telling anything about the intrinsic merit of a single variable (in terms of physics meaning for instance), but is rather a measure of its importance for this particular training.

Another advantage of this approach is that it can be applied on the validation set as well. Variables that are important on the training set but not on the validation set may be a source of overfitting.

As with MDI however, correlations may hide the intrinsic performance of a variable. If two variables are correlated and only one is shuffled, the proper information is still accessible, giving a lower importance to both. Once again, interpreting variable rankings must be done with care.

3.4.4. *Choosing variables*

It may sound obvious that only well discriminating variables should be used as input features to the decision tree training. It is nevertheless not trivial to achieve: variables are often correlated, they come in large numbers, and can be more or less discriminating in various regions of the input-feature phase space. The decision tree will

isolate subregions, whose properties are not readily available when measuring any kind of discrimination in the full training set.

Brute force is a possibility: with a limited number of N features, train all possible combinations of N , $N - 1$, etc., variables, and pick the best one according to some metric (see Sec. 2.5). In reality this becomes quickly impractical.

Instead, a commonly used approach in high-energy physics is backward elimination [21], which starts from the full list of N variables used to train a tree (T_N). Then train all decision trees with $N - 1$ variables and keep the best performing one on the validation set (T_{N-1}). Starting from these $N - 1$ variables, train all decision trees with $N - 2$ variables to build T_{N-2} , and so on. Usually the performance of tree T_k will decrease with k , and it is up to the analyzer to decide how much performance to lose compared to getting a simpler (possibly more robust) tree. This is the usual trade-off of cost and complexity.

The selection can also be done in reverse, starting from $k = 1$ variable, training all trees with $k + 1$ variables, keeping the best one on the validation set and moving to $k + 2$ variables, until $k = N$ (forward greedy selection [21]). The advantage is that one can stop adding variables once the performance curve seems to saturate. It is on the other hand not equivalent to backward elimination, as it may miss powerful variable combinations.

It can be tempting to train a tree with many variables and then remove the lowest ranked. Although quicker, it will most certainly be suboptimal because of the shortcomings of such rankings, as described in Secs. 3.4.2 and 3.4.3. The ranking is only relevant to the corresponding tree, and as soon as one of the variables is removed the others may be reshuffled.

3.5. *Limitations*

Despite all the nice features presented above, decision trees are known to be relatively unstable. If trees are too optimized for the training sample, they may not generalize very well to unknown events, as they would depend on the training sample (see Sec. 3.5.1). This can be mitigated with pruning, described in Sec. 3.5.2. Combining several

classifiers can also improve the overall performance, as shown in Sec. 3.5.3.

3.5.1. *Training sample composition*

A small change in the training sample can lead to drastically different tree structures (high variance), rendering the physics interpretation a bit less straightforward. As such, a decision tree is not stable, where stability means that a slight change of the inputs does not change much the output [21]. For sufficiently large training samples, the performance of these different trees will be equivalent, but on small training samples variations can be very large. This does not give too much confidence in the result.

Moreover a decision tree output is by nature discrete, limited by the purities of all leaves in the tree. To decrease the discontinuities the tree size and complexity has to increase, which may not be desirable or even possible. Then the tendency is to have spikes in the output distribution at specific purity values, or even two delta functions at ± 1 if using a binary answer rather than the purity output.

3.5.2. *Pruning a tree*

When growing a tree, each node contains fewer and fewer events, leading to an increase of the statistical uncertainty on each new split. The tree will tend to become more and more specialized, focusing on properties of the training sample that may not reflect the expected result, had there been infinite statistics to train on. Its variance increases.

A first approach to mitigate this effect and keep the variance under control, sometimes referred to as pre-pruning, has already been described in Sec. 3, using stopping conditions. The limitation is that requiring too big a minimum leaf size or too much of an improvement may prevent further splitting that could be very beneficial later on.

Another approach consists in building a very large tree and then cutting irrelevant branches (which target too closely the training sample and would not generalize well) by turning an internal node

and all its descendants into a leaf, removing the corresponding subtree. This is post-pruning, or simply pruning.

There are many different pruning algorithms available. Expected error pruning [22] starts from a fully grown tree and compares the expected error of a node to the weighted sum of expected errors from its children. If the expected error of the node is less than that of the children, then the node is pruned. This does not require a separate pruning sample. With reduced error pruning [22] the misclassification rate on a pruning sample for the full tree is compared to the misclassification rate when a node is turned into a leaf. If the simplified tree has better performance, the subtree is pruned. Finally, cost-complexity pruning is part of the CART algorithm [6] and the most used. Starting from a fully grown tree, the cost-complexity is computed as the sum of misclassification rate and a term proportional to the number of nodes in the tree (the complexity part, penalizing larger trees). A sequence of decreasing cost-complexity subtrees is generated, and their misclassification rate on the pruning sample is computed. It will first decrease, and then go through a minimum before increasing again. The optimally pruned tree is the one corresponding to the minimum.

It should be noted that the best pruned tree may not be optimal or necessary when part of a forest of trees, such as those introduced in the following sections.

3.5.3. *Ensemble learning*

Pruning is helpful in maximizing the generalization potential of a single decision tree. It nevertheless does not address other shortcomings of trees like the discrete output or lack of stability. A way out is to proceed with averaging several trees, with the added potential bonus that the discriminating power may increase. Such approaches belong to the general theoretical framework of ensemble learning [23]. Many averaging techniques have been developed. Bagging, boosting and random forests are such techniques and will be described in the following sections.

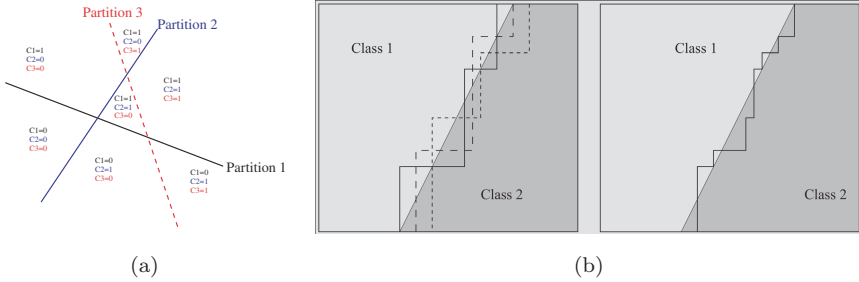


Fig. 6. (a) Description of 2D space combining three discriminants. (b) Three separate decision trees and their combination [24].

The power of ensemble learning resides in the much richer description of the input patterns when using several classifiers simultaneously. It is applicable to other machine learning techniques than decision trees. As shown in the example of Fig. 6(a) in a simple 2D case, a classifier may split the space in two (partitions 1/2/3), but three classifiers each doing this can possibly give more complete information about seven regions, each region being represented by three numbers (C1/C2/C3). When all three classifiers give the same answer, the confidence increases. Using decision trees as in Fig. 6(b), three simple decision trees give a crude separation of classes 1 and 2, while averaging them produces a decision contour that is much closer to the actual class separation.

4. Boosted Decision Trees

As will be shown in this section, the boosting algorithm has turned into a very successful way of improving the performance of any type of classifier, not only decision trees. After a short history of boosting in Sec. 4.1, the generic algorithm is presented in Sec. 4.2 and specific implementations (AdaBoost and gradient boosting) are described in Secs. 4.3 and 4.4. Boosting is illustrated with a few examples in Sec. 4.5. Other boosting implementations are shown in Sec. 4.6. The use of boosting for regression rather than classification is presented in Sec. 4.7. Finally the application of boosted decision trees in

high-energy physics, where it is so far the machine learning algorithm of choice, is illustrated in Sec. 4.8.

4.1. *Introduction*

The first provable algorithm of boosting was proposed in 1990 [25]. It worked in the following way:

- train a classifier T_1 on a sample of N events;
- train T_2 on a new sample with N events, half of which were misclassified by T_1 ;
- build T_3 on events where T_1 and T_2 disagree.

The boosted classifier was defined as a majority vote on the outputs of T_1 , T_2 and T_3 .

Following up on this idea boosting by majority [26] was introduced in 1995. It consisted in combining many learners with a fixed error rate. This was an impractical prerequisite for a viable automated algorithm, but was a stepping stone to the first functional boosting algorithm, called AdaBoost [27].

Boosting, and in particular boosted decision trees, have become increasingly popular in high-energy physics and are extensively used in physics analyses and object identification at the Tevatron and the LHC (see Sec. 4.8 for a few examples).

4.2. *Boosting algorithm*

It is hard to make a very good discriminant, but relatively easy to make simple ones which are certainly more error-prone (high bias) but are still performing at least marginally better than random guessing. Such discriminants are called weak classifiers. The goal of boosting is to combine such weak classifiers into a new, more stable one, with a smaller error rate (with lower bias than the individual classifiers) and better performance.

Consider a training sample \mathbb{T}_k containing N_k events. The i th event is associated with a weight w_i^k , a vector of discriminating variables \vec{x}_i and a class label $y_i = +1$ for signal, -1 for background. The pseudocode for a generic boosting algorithm is:

```

Initialize  $\mathbb{T}_1$ 
for  $k$  in  $1..N_{\text{tree}}$ 
    train classifier  $T_k$  on  $\mathbb{T}_k$ 
    assign weight  $\alpha_k$  to  $T_k$ 
    modify  $\mathbb{T}_k$  into  $\mathbb{T}_{k+1}$ 

```

The boosted output is some function $F(T_1, \dots, T_{N_{\text{tree}}})$, typically a weighted average:

$$F(i) = \sum_{k=1}^{N_{\text{tree}}} \alpha_k T_k(\vec{x}_i).$$

Thanks to this averaging, the output becomes quasi-continuous, mitigating one of the limitations of single decision trees (see Sec. 3.5.1).

Note that in this process, once a particular tree is trained it is never modified, but just added to the mix. This is a different approach from, e.g., neural networks, in which the same weights are repeatedly updated over epochs to converge towards the final classifier.

4.3. *AdaBoost*

One particularly successful implementation of the boosting algorithm is AdaBoost [27]. AdaBoost stands for adaptive boosting, referring to the fact that the learning procedure adjusts itself to the training data in order to classify it better. There are many variations for the actual implementation, and it is the most common boosting algorithm. It typically leads to better results than without boosting, up to the Bayes limit as will be seen later.

An actual implementation of the AdaBoost algorithm works as follows. After having built tree T_k , events in the training sample \mathbb{T}_k that are misclassified by T_k should be checked, hence defining the misclassification rate $R(T_k)$. In order to ease the math, let us introduce some notations. Define $\mathbb{I} : X \rightarrow \mathbb{I}(X)$ such that $\mathbb{I}(X) = 1$ if statement X is true, and 0 otherwise. A function can now be defined that tells whether an event is misclassified by T_k . In the decision tree

output convention of returning only $\{\pm 1\}$ it gives:

$$\text{isMisclassified}_k(i) = \mathbb{I}(y_i \times T_k(i) \leq 0),$$

while in the purity output convention (with a critical purity of 0.5) it leads to:

$$\text{isMisclassified}_k(i) = \mathbb{I}(y_i \times (T_k(i) - 0.5) \leq 0).$$

The misclassification rate is now:

$$R(T_k) = \varepsilon_k = \frac{\sum_{i=1}^{N_k} w_i^k \times \text{isMisclassified}_k(i)}{\sum_{i=1}^{N_k} w_i^k}.$$

This misclassification rate can be used to derive a weight associated to tree T_k :

$$\alpha_k = \beta \times \ln \frac{1 - \varepsilon_k}{\varepsilon_k},$$

where β is a free parameter to adjust the strength of boosting (set to one in the original algorithm). Similarly to the naming convention of other machine learning algorithms, it can be seen as a learning rate or shrinkage coefficient and drives how aggressive boosting should be.

The core of the AdaBoost algorithm resides in the following step: each event in \mathbb{T}_k has its weight changed in order to create a new sample \mathbb{T}_{k+1} such that:

$$w_i^k \rightarrow w_i^{k+1} = w_i^k \times e^{\alpha_k \cdot \text{isMisclassified}_k(i)}.$$

This means that properly classified events are unchanged from \mathbb{T}_k to \mathbb{T}_{k+1} , while misclassified events see their weight increased by a factor e^{α_k} . The next tree T_{k+1} is then trained on the \mathbb{T}_{k+1} sample. This next tree will therefore see a different sample composition with more weight on previously misclassified events, and will therefore try harder to classify properly difficult events that tree T_k failed to identify correctly, while leaving alone those events that previous iterations can handle properly. The final AdaBoost result for event i is

$$T(i) = \frac{1}{\sum_{k=1}^{N_{\text{tree}}} \alpha_k} \sum_{k=1}^{N_{\text{tree}}} \alpha_k T_k(i).$$

As an example, assume for simplicity the case $\beta = 1$. A not-so-good classifier, with a misclassification rate $\varepsilon = 40\%$ would have a corresponding $\alpha = \ln \frac{1-0.4}{0.4} = 0.4$. All misclassified events would therefore get their weight multiplied by $e^{0.4} = 1.5$, and the next tree will have to work a bit harder on these events. Now consider a good classifier with an error rate $\varepsilon = 5\%$ and $\alpha = \ln \frac{1-0.05}{0.05} = 2.9$. Misclassified events get a boost of $e^{2.9} = 19$ and will contribute decisively to the structure of the next tree! This shows that being failed by a good classifier brings a big penalty.

It can be shown [28] that the misclassification rate ε of the boosted result on the training sample is bounded from above:

$$\varepsilon \leq \prod_{k=1}^{N_{\text{tree}}} 2\sqrt{\varepsilon_k(1 - \varepsilon_k)}.$$

If each tree has $\varepsilon_k \neq 0.5$, that is to say, if it does better than random guessing, then the conclusion is quite remarkable: the error rate falls to zero for a sufficiently large N_{tree} . A corollary is that the training data is overfit.

Overtraining is usually regarded as a negative feature. Does this mean that boosted decision trees are doomed because they are too powerful on the training sample? Not really. As shown in Sec. 2.6 what matters most is not the error rate on the training sample, but rather the error rate on the testing sample. In the case of Fig. 3(a) or Fig. 3(b) boosting should stop when the minimum is reached (early stopping). It has however been routinely observed [29–31] that boosted decision trees often do not go through such a minimum, but rather tend towards a plateau in testing error (see Fig. 3(c)). Boosting could be stopped after having reached this plateau.

In a typical high-energy physics problem, the error rate may not even be what should be optimized. A good figure of merit on the testing sample would rather be the significance. Figure 7(a) illustrates this behavior, showing how the significance saturates with an increasing number of boosting cycles. Arguably one could stop before the end and save resources, but at least the performance does not deteriorate with increasing boosting.

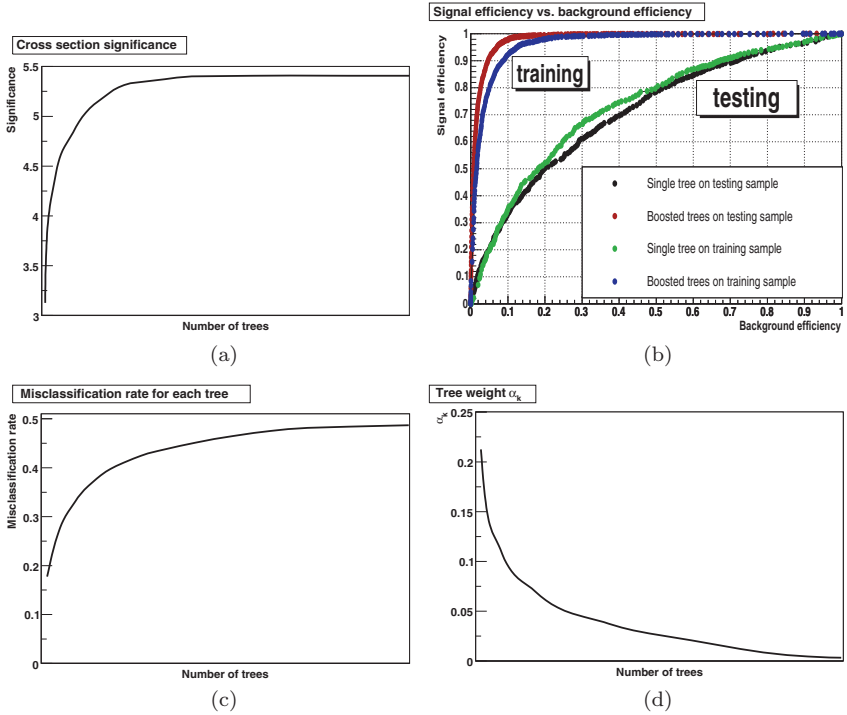


Fig. 7. Behavior of boosting. (a) Significance as a function of the number of boosted trees. (b) Signal efficiency vs. background efficiency for single and boosted decision trees, on the training and testing samples. (c) Misclassification rate of each tree as a function of the number of boosted trees. (d) Weight of each tree as a function of the number of boosted trees.

Another typical curve to optimize is the signal efficiency vs. the background efficiency (the ROC curve, see Sec. 2.5.1). Figure 7(b) clearly exemplifies this interesting property of boosted decision trees. The performance is clearly better on the training sample than on the testing sample (the training curves are getting very close to the upper left corner of perfect separation), with a single tree or with boosting, a clear sign of overtraining. But the boosted tree is still performing better than the single tree on the testing sample, proof that it does learn something more than memorizing the training sample.

No clear explanation has emerged as to why boosting leads to such features, with typically no loss of generalization performance

due to overtraining, but some ideas have come up. It may have to do with the fact that during the boosting sequence, the first tree is the best while the others are successive minor corrections, which are given smaller weights. This is shown in Figs. 7(c) and 7(d), where the misclassification rate of each new tree separately is actually increasing, while the corresponding tree weight is decreasing. This is not surprising: during boosting the successive trees are specializing on specific event categories, and can therefore not perform as well on other events. So the trees that lead to a perfect fit of the training data are contributing very little to the final boosted decision tree output on the testing sample. When boosting decision trees, the last tree is not an evolution of the first one that performs better, quite the contrary. The first tree is typically the best, while others bring dedicated help for misclassified events. The power of boosting does not rely in the last tree in the sequence, but rather in combining a suite of trees that focus on different events.

A probabilistic interpretation of AdaBoost was proposed [31] which gives some insight into the performance of boosted decision trees. It can be shown that for a boosted output T flexible enough:

$$e^{T(i)} = \frac{p(S|i)}{p(B|i)}.$$

This means that the AdaBoost algorithm will tend towards the Bayes classifier, the maximum reachable separation.

Finally, AdaBoost performance and its tendency to generalize well despite matching very closely the training data (to the extent that in many documented cases, to keep boosting even after the training error has reached zero still improves the performance on the testing sample [29], in the interpolation regime [15]) have been qualitatively understood with the margins explanation [29, 32]. A classifier can be more sure of some predictions than of others (recall Fig. 6(a)), and could then generalize better. By boosting, AdaBoost tends to increase the margins on the training set, even after reaching zero training error. For each event, the margin accounts for the separability between classes, measured by the proportion of trees that misclassify each event. For event x with truth label y , the margin

$y \times T(x)$ for boosted decision tree T is

$$\begin{aligned} y \times T(x) &= \frac{y}{\sum_{k=1}^{N_{\text{tree}}} \alpha_k} \sum_{k=1}^{N_{\text{tree}}} \alpha_k T_k(x) \\ &= \frac{1}{\sum_{k=1}^{N_{\text{tree}}} \alpha_k} \left(\sum_{k:y=T_k(x)} \alpha_k - \sum_{k:y \neq T_k(x)} \alpha_k \right), \end{aligned}$$

that is, the difference between the weights of single trees that classify x correctly and the weights of trees that misclassify x . Boosting more means adding small corrections that tend to increase the margin for each event. This increases the confidence in the prediction, more likely to be correct. It makes a link with support vector machines [33], although this did not bring great insights to improve AdaBoost in the end.

This shortcoming suggests that there may be other explanations, as discussed in [15], focusing on the interpolation regime when the training error has already reached zero but boosting further still leads to testing error improvement (better generalization). The combination of large trees focusing on extremely local neighborhoods of the training dataset and averaging over a large number of trees seems to prevent overfitting efficiently. This has been interpreted in the more general framework of double descent risk curve [16]. With boosting, the interpolating regime behavior (see Fig. 3(d)) may kick in even before the interpolating threshold, possibly explaining why typical boosted decision tree training curves look like Fig. 3(c).

4.4. Gradient boosting

While trying to understand how AdaBoost and other boosting algorithms work, they were originally recast in the statistical framework of arcing algorithms (an acronym for adaptive reweighting and combining) [34, 35]. At each step, a weighted minimization is performed followed by a recomputation of the classifier and weighted input. This was further developed to become gradient boosting [30]. Boosting is formulated as a numerical optimization problem, trying to minimize

the loss function by adding trees using a gradient descent procedure rather than giving a higher weight to misclassified events.

Formally, consider a model F built iteratively, its imperfect instance at step k being F_k . F_k is therefore an approximation of the best possible model (in some cases $F_k(x) \neq y$), which is to be improved at the next iteration. This is achieved by adding a new component h_k such that:

$$F_{k+1}(x) = F_k(x) + h_k(x) = y,$$

or equivalently:

$$h_k(x) = y - F_k(x).$$

Rather than training F_{k+1} a new classifier can be trained to fit the residual $y - F_k(x)$, which corresponds to the part that the current model F_k cannot treat correctly. If $F_{k+1}(x)$ is still not satisfactory, new iterations can be fitted.

The link with gradient descent is explicit when considering the particular case of the mean squared error (MSE) loss function (a typical case for regression problems, see Sec. 4.7):

$$L_{\text{MSE}}(x, y) = \frac{1}{2} (y - F_k(x))^2.$$

Minimizing the loss $J = \sum_i L_{\text{MSE}}(x_i, y_i)$ by adjusting all $F_k(x_i)$ leads to:

$$\frac{\partial J}{\partial F_k(x_i)} = \frac{\partial L_{\text{MSE}}(x_i, y_i)}{\partial F_k(x_i)} = F_k(x_i) - y_i.$$

Residuals can therefore be interpreted as negative gradients:

$$h_k(x_i) = y_i - F_k(x_i) = -\frac{\partial J}{\partial F_k(x_i)}.$$

The concept can be generalized to any differentiable loss function instead of MSE. For instance AdaBoost corresponds to an exponential loss $e^{-F_k(x)y}$.

There are several variants of gradient boosting algorithms on the market. Techniques presented in Sec. 5 with subsampling of the training set and tree parameters can be used (in particular a bagging-like

approach without replacement), leading to stochastic gradient boosting [36]. These regularization techniques help prevent overfitting.

4.5. Boosting examples

The examples of this section illustrate typical behaviors of boosted decision trees.

4.5.1. The XOR problem

The XOR problem is a small version of the checkerboard, illustrated in Fig. 8. With enough statistics (Figs. 8(a) and 8(c)), even a single

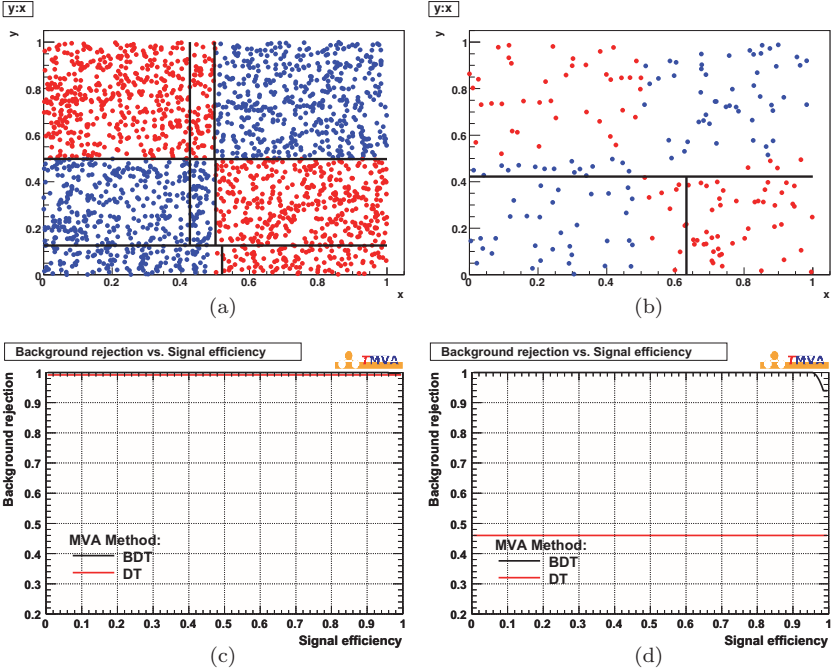


Fig. 8. The XOR problem. Signal is in blue, background in red. The left column (a and c) uses sufficient statistics, while the right column has a limited number of training events. The top plots (a and b) show the signal and background distributions as well as the criteria used by the first decision tree. Bottom plots (c, d) illustrate the background rejection vs. signal efficiency curves for the first decision tree (red) and for the boosted decision trees (black), all run on the same testing events.

tree is already able to find more or less the optimal separation, so boosting cannot actually do much better.

The exercise can be repeated, this time with limited statistics (Figs. 8(b) and 8(d)). Now a single tree is not doing such a good job anymore. Boosted decision trees, on the other hand, are doing almost as well as with full statistics, separating almost perfectly signal and background. This illustrates very clearly how the combination of weak classifiers (see for instance the lousy performance of the first tree) can generate a high-performance discriminant with a boosting algorithm.

4.5.2. Number of trees and overtraining

This example uses a highly correlated dataset, shown in Fig. 9(a).

Figure 9(b) compares the performance of a single decision tree and boosted decision trees with an increasing number of trees (from 5 to 400). All other parameters are kept to their default value in the TMVA package [37]. The performance of the single tree is not so good, as expected since the default parameters make it very small, with a depth of 3 (it should be noted that a single bigger tree could solve this problem easily). Increasing the number of trees improves

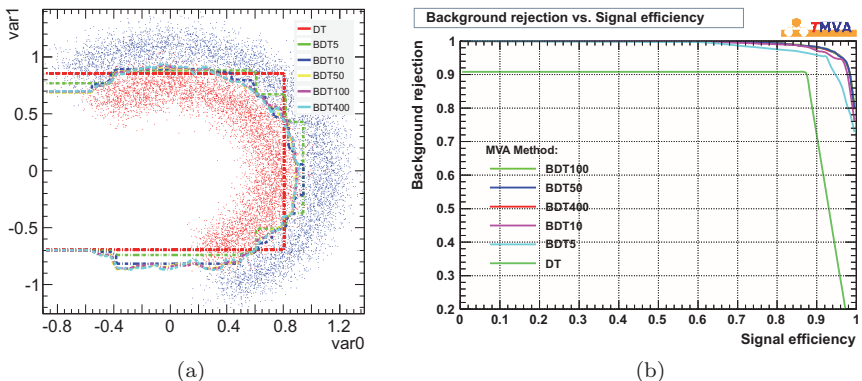


Fig. 9. (a) 2D dataset and decision contour corresponding to several discriminants. (b) Background rejection vs. signal efficiency curves for a single decision tree (dark green) and boosted decision trees with an increasing number of trees (5–400).

the performance until it saturates in the high-background rejection and high-signal efficiency corner. Adding more trees does not seem to degrade the performance, the curve stays in the optimal corner. Looking at the contours in Fig. 9(a) it wiggles a little for larger boosted decision trees, as they tend to pick up features of the training sample. This is overtraining.

Another sign of overtraining also appears in Fig. 10, showing the output of the various boosted decision trees for signal and background, both on the training and testing samples: larger boosted decision trees tend to show differences between the two samples (as quantified by a Kolmogorov–Smirnov (KS) test in the figures, especially Fig. 10(f)), as they adjust to peculiarities of the training sample that are not found in an independent testing sample. The output acquires a “better” shape with more trees, really becoming quasi-continuous, which would allow to cut at a precise efficiency or rejection.

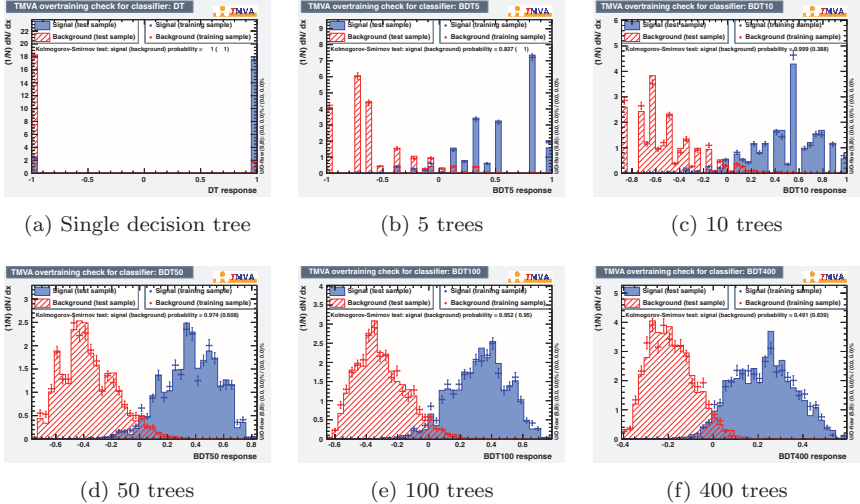


Fig. 10. Comparison of the output on training (markers) and testing (histograms) signal (blue) and background (red) samples for boosted decision trees with 1, 5, 10, 50, 100 and 400 trees (from top left to bottom right). The Kolmogorov–Smirnov test quantifies the (dis)agreement between training and testing outputs.

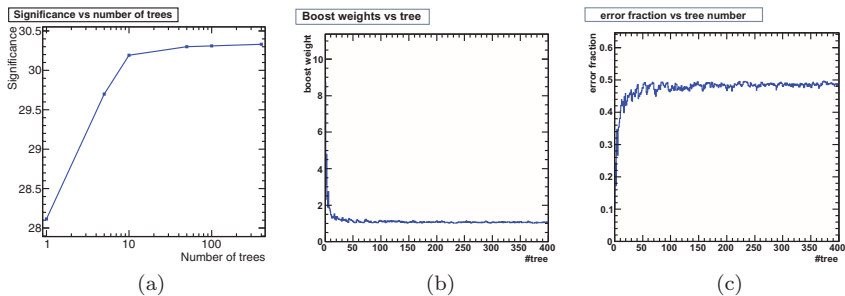


Fig. 11. (a) Maximum significance of all boosted decision trees. (b) Boost weight of each tree. (c) Error fraction of each tree (0.5 means random guessing).

Both figures do exhibit clear signs of overtraining, but is it really an issue? As mentioned before (see Sec. 2.6) what really matters in the end is the performance in data analysis and on the testing sample. One way to evaluate this is to compute the maximum significance $s/\sqrt{s+b}$ (see Sec. 2.5.2). It is shown in Fig. 11(a) for the same boosted decision trees as shown in Fig. 10, with increasing number of trees. The best significance is actually obtained with the 400-tree boosted decision tree, following what was described at the end of Sec. 4.3. To be fair, the performance is very similar already with 10 trees. Now, comparing the outputs in Fig. 10, if interested in a smoother result, 10 trees might not be enough, but 50 would probably do, without the overhead of eight times more trees. Such a choice should in any case not be made based on overtraining statements comparing performance on the training and testing samples (as some are tempted to do, seeing an increasing disagreement, quantified by the KS test, between outputs on the training and testing samples), but rather on final expected physics performance (the final number of the analysis, for instance the significance from the complete statistical analysis, possibly including systematic uncertainties). Boosted decision trees are often in the situation described in Fig. 3(c), meaning that their performance is not decreasing when boosting longer, even as the discrepancy in performance between train and test keeps increasing.

This example also illustrates the performance of each tree in a boosting sequence. Figure 11(b) shows the rapid decrease of the

weight α_k of each tree, while at the same time the corresponding misclassification rate ε_k of each individual tree increases rapidly towards just below 50%, that is, random guessing (Fig. 11(c)). It confirms that the best trees are the first ones, while the others are only minor corrections.

4.6. Other boosting algorithms

AdaBoost is but one of many boosting algorithms. It is also referred to as discrete AdaBoost to distinguish it from other AdaBoost flavors. The Real AdaBoost algorithm [31] defines each decision tree output as follows:

$$T_k(i) = 0.5 \times \ln \frac{p_k(i)}{1 - p_k(i)},$$

where $p_k(i)$ is the purity of the leaf on which event i falls. Events are reweighted as:

$$w_i^k \rightarrow w_i^{k+1} = w_i^k \times e^{-y_i T_k(i)}$$

and the boosted result is $T(i) = \sum_{k=1}^{N_{\text{tree}}} T_k(i)$. Gentle AdaBoost and LogitBoost (with a logistic function) [31] are other variations.

ε -Boost, also called shrinkage [30], consists in reweighting misclassified events by a fixed factor $e^{2\varepsilon}$ rather than the tree-dependent α_k factor of AdaBoost. ε -LogitBoost [31] is reweighting them with a logistic function $\frac{e^{-y_i T_k(i)}}{1 + e^{-y_i T_k(i)}}$. ε -HingeBoost [2] is only dealing with misclassified events:

$$w_i^k \rightarrow w_i^{k+1} = \mathbb{I}(y_i \times T_k(i) \leq 0).$$

Finally, the adaptive version of the “boost by majority” [26] algorithm is called BrownBoost [38]. It works in the limit where each boosting iteration makes an infinitesimally small contribution to the total result, modeling this limit with the differential equations that govern Brownian motion.

4.7. Boosted regression trees

From their very introduction [6], trees have been considered for classification (decision trees) and for regression (regression trees), where

instead of identifying “signal-like” or “background-like” regions of phase space, tree leaves each contain a single real value supposed to approach the target function.

During tree building for regression, the maximization of the decrease of impurity in decision trees is replaced by the reduction of the standard deviation or of the mean squared error:

$$d(t) = \frac{1}{N_t} \sum_{N_t} (y - \hat{y}_t)^2,$$

for a node t with N_t events, regression target y of each event in the node and mean value \hat{y}_t of regression targets of all events in the node. Another typical choice for d is the mean absolute error:

$$\frac{1}{N_t} \sum_{N_t} |y - \text{median}(y)_t|.$$

Constructing a regression tree is about finding the attribute that return the highest reduction in d (i.e. the most homogeneous nodes) when going from node t to nodes t_P and t_F (see Sec. 3.3 for notations):

$$\Delta d(S, t) = d(t) - p_P \cdot d(t_P) - p_F \cdot d(t_F).$$

The splitting stops when nodes become too small or when their internal variation is sufficiently small. The regression tree output is the mean (or median if using the mean absolute error) value of the training events in the corresponding (leaf) node. So a regression tree partitions the feature space of input variables into hyperrectangles and then fits a constant inside each box.

When boosting regression trees, there are no longer properly and wrongly classified events, so the misclassification rate cannot be computed to reweight events. Instead the average loss $\langle L^k \rangle$ after the k th tree is computed over the training sample, and the boosting quantity $\beta_k = \langle L^k \rangle / (1 - \langle L^k \rangle)$ is derived. The reweighting of events is then computed based on their individual loss $L^k(i)$:

$$w_i^k \rightarrow w_i^{k+1} = w_i^k \times \beta_k^{1-L^k(i)}.$$

The training process is then similar to that of boosted decision trees, and the final prediction of the fitted value is the weighted average of all tree outputs.

4.8. *Boosted decision trees in high-energy physics*

Boosted decision trees have become very popular in high-energy physics. A few usage examples are presented in Sec. 4.8.1. Their proper usage also means addressing issues linked to systematic uncertainties, as reported in Sec. 4.8.2.

4.8.1. *Use cases*

The MiniBooNe experiment at Fermilab, searching for neutrino oscillations, was the first in the field to compare the performance of different boosting algorithms and artificial neural networks for analysis and particle identification [1, 2], on Monte Carlo samples. Trees with up to 120 variables were tested, with different boosting algorithms and up to thousands of trees. These studies introduced boosted decision trees in the particle physics world.

The concept of boosted decision trees was picked up by the D0 experiment at Fermilab, leading to the first evidence (and then observation) of single top quark production in Tevatron data [3, 4]. Among the 49 variables used, some had very similar definitions (like the scalar sum of transverse momentum of various jets), which was beneficial as not all of them suffer from the same mismeasurements on an event-by-event basis. Boosted decision trees happened to perform slightly better than two other techniques used: the matrix element calculation and Bayesian neural networks. Without such advanced techniques, the signal could not have been seen with the dataset available at the time: the total uncertainty on the model prediction was much larger than the expected signal, as illustrated in Fig. 12(a). This also means that no single distribution (apart from the boosted decision tree output shown in Fig. 12(b)) could really show the new observed process, leading to scepticism in the community (“I want to see a mass peak!” is a common argument, reflecting on the fact that people are more confident in the result if they

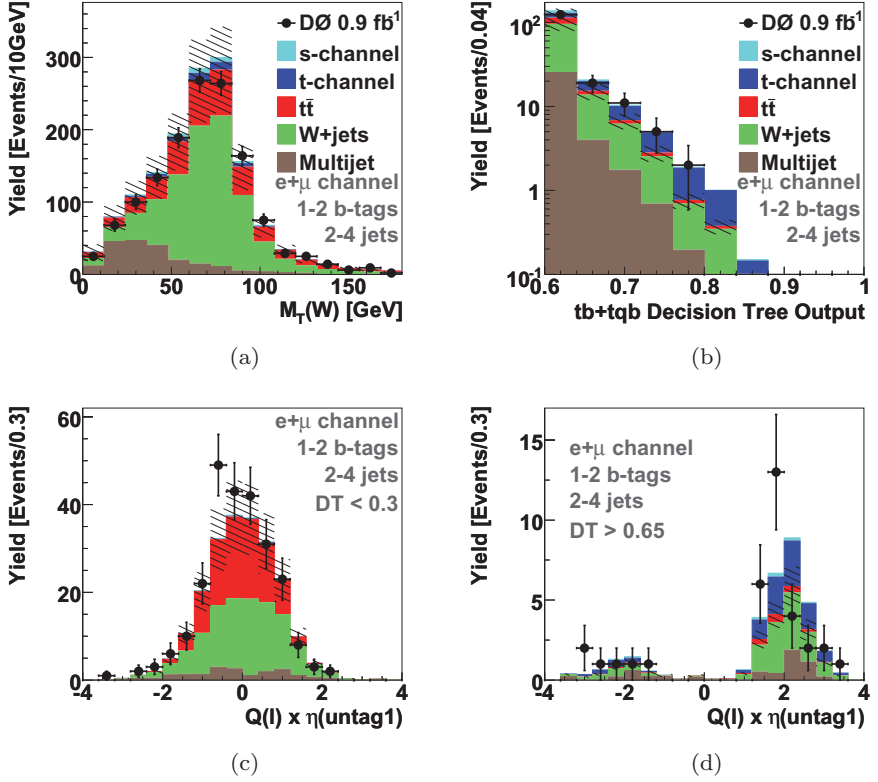


Fig. 12. Usage of boosted decision trees in physics analysis [4]. (a) A discriminating variable, with uncertainty larger than the expected signal (in blue). (b) The boosted decision tree output with much smaller uncertainty. (c, d) Discriminating variable when selecting only events with low (high) boosted decision tree output, showing background (signal)-like shape.

can see the signal in a physical distribution). Various cross-checks were performed to increase the degree of belief in the final outcome (removing top-quark-mass-related variables during training, validating the description of the boosted decision tree output in regions depleted in signal, analyzing the shape of other variables after selecting low- or high-boosted decision tree output events enriched in background or signal events as shown in Fig. 12(c) and Fig. 12(d), respectively, etc.).

Since then, boosted decision trees have become a bread and butter technique in high-energy physics and are extensively used in physics analyses (to extract their tiny signal from large backgrounds or distinguish between different signals) and object identification at the Tevatron or the LHC. In the ATLAS experiment τ -lepton identification [39] and flavor tagging [7] used boosted decision trees in Run 2, and the τ -lepton energy is estimated with boosted regression trees. The LHCb trigger was reoptimized, comparing the performance of several tree-based algorithms to neural networks [40], while their muon identification performance for the Run 3 of the LHC will profit from improvements thanks to gradient boosting [41].

The latest result has just been published at the time of writing, reporting the first evidence for $t\bar{t}t\bar{t}$ production in ATLAS [8], shown in Fig. 1(b). The one analysis using the most boosted decision trees is probably the observation of the diphoton decay of the Higgs boson by the CMS experiment [42]. The diphoton vertex is selected with a boosted decision tree, while another one estimates, event-by-event, the probability for the vertex assignment to be within 10 mm of the diphoton interaction point. Photons are identified with a boosted decision tree, and their energy is corrected with a boosted regression tree that provides the energy and its associated uncertainty. Finally several boosted decision trees are used to select the various signal regions and extract signal from these different categories.

Beyond object identification and calibration, and final discriminant in physics analyses, boosted decision trees can also be used to reduce the number of potential object combinations in order to find the correct match between the observed objects in the detector and their probable source of production. Such a “reconstruction BDT” was used to look for the associated production of a Higgs boson and a pair of top quarks, $t\bar{t}H(b\bar{b})$ [43].

Lately there is a tendency towards deep neural networks and their many flavors to replace boosted decision trees in the various stages of analysis [44, 45]. Boosted decision trees nevertheless remain a favorite in high-energy physics, for their ease of use, high-performance out-of-the-box, limited required tuning of hyperparameters and resilience against overtraining.

4.8.2. *Systematic uncertainties*

There is an *a priori*, especially among physicists not very familiar with machine learning techniques, to distrust their output because they are not a measurable quantity with a physical meaning like an invariant mass. They are indeed complex variables, but so are for instance energy quantities for reconstructed particles in the detector. Uncertainties on such “basic” variables are typically evaluated by varying the value of a requirement, changing the calibration of objects that go into the variable, etc. The boosted decision tree output (or of any such multivariate technique) is no different: its inputs can be varied according to their known uncertainties (for instance varying the jet energy scale will have a correlated impact on all discriminating variables that depend on jets) and their effect propagated through the boosted decision tree (the shifted inputs will lead to a different boosted decision tree output), to see how much these changes impact the analysis. This gives the size of the uncertainty on the multivariate discriminant output.

That being said, the Peter Parker principle applies: “With great power comes great responsibility”. Boosted decision trees are very powerful, and will target small areas of phase space where potentially not all known systematic uncertainties are strictly valid. Then extra uncertainties may be needed, not so much on the technique itself but rather due to the fact that it extracts information from less well-known regions.

Usually boosted decision trees are trained on the nominal Monte Carlo samples and are therefore completely oblivious to the effect of systematic uncertainties. This could lead to bad results once they are introduced, if the boosted decision trees are sensitive to them, and when applied on real data. One way to possibly mitigate this effect is with one form of data augmentation, training the boosted decision trees on a mixture of nominal and systematically shifted events, hence increasing the training statistics and allowing the boosted decision trees to see other events than the nominal ones during training to learn their features. The nominal performance should decrease,

but with the hope that systematic uncertainties will have less of an impact on the final measurement. Experience with this approach is inconclusive. If the physics model is not properly describing the real data, then the performance will also be affected. It can be partially addressed with domain adaptation [5] (as described elsewhere in this book).

5. Other Averaging Techniques

As mentioned in Sec. 3.5.3 the key to improving a single decision tree performance and stability is averaging. Other techniques than boosting exist, some of which are briefly described below. As with boosting, statistical perturbations are introduced to randomize the training sample, hence increasing the predictive power of the ensemble of trees.

Bagging (Bootstrap AGGREGatING) was proposed in [46]. It consists in training trees on different bootstrap samples drawn randomly with replacement from the training sample. Events that are not picked for the bootstrap sample form an “out-of-bag” validation sample. The bagged output is the simple average of all such trees, with a reduced variance compared to individual trees.

Random forests is bagging with an extra level of randomization [20]. Before splitting a node, only a random subset of discriminating variables is considered. The fraction can vary for each split for yet another level of randomization.

Trimming is not exactly an averaging technique per se but can be used in conjunction with another technique, in particular boosting, to speed up the training process. After some boosting cycles, it is possible that very few events with very high weight are making up most of the total training sample weight. Events with very small weights may be ignored, hence introducing again some minor statistical perturbations and speeding up the training. ε -HingeBoost is such an algorithm (see Sec. 4.6).

6. Software

Many implementations of decision trees exist on the market. Some of them, all open source, are briefly presented below.

The most popular in high-energy physics is TMVA [37], integrated into ROOT. It includes single decision trees, boosted trees with AdaBoost and gradient boost, bagging and random forests. Being part of ROOT it is very straightforward to use within usual analysis frameworks, both in C++ and Python. It includes tools for data preparation and makes it simple to compare performance between many algorithms, not only tree-based ones. Already mentioned Refs. [7, 8, 39, 42, 43] are but a few examples of TMVA usage in the field.

Another implementation has gained visibility in high-energy physics: XGBoost [47]. It entered the field after receiving to special HEP meets ML award during the Higgs boson machine learning challenge (HiggsML) hosted by Kaggle [11] (described in Chapter 20). It features a high-performing, scalable gradient boosting implementation, capable of using GPU and large cluster parallelization. Instead of the greedy algorithm described in Sec. 3.1, the authors developed an approximate algorithm that proposes candidate splitting points according to percentiles of the input variables, and then maps the variables into buckets according to these splits to find the best solution. Many analyses at the LHC are now using it (see, for instance, [48]).

Other implementations have lower usage in high-energy physics so far while being used in other fields. LightGBM (light gradient boosting machine [49]), originally developed by Microsoft, is competing with XGBoost in speed, scalability and performance. It builds trees in a very different way from what was presented in this chapter, with a histogram-based decision tree learning algorithm. Scikit-learn [50] is a very popular machine learning framework with several tree-related implementations and utilities for data preparation. Finally CatBoost [51] is a new gradient boosting implementation from Yandex used in commercial services as well as in high-energy physics, for instance in LHCb [41].

7. Conclusion

This chapter introduced what decision trees are and how to construct them, as a powerful multivariate extension of a cut-based analysis. Advantages are numerous: their training is fast, they lead to human-readable results (not black boxes) with possible interpretation by a physicist, can deal easily with all sorts of variables and with many of them, with in the end relatively few parameters.

Decision trees are, however, not perfect and suffer from the piece-wise nature of their output and a high sensitivity to the content of the training sample. These shortcomings are for a large part addressed by averaging the results of several trees, each built after introducing some statistical perturbation in the training sample. Among the most popular such techniques, boosting (and its AdaBoost and gradient boost incarnations) was described in detail, providing ideas as to why it seems to be performing so well while being very resilient against overtraining. Other averaging techniques were briefly presented.

Boosted decision trees have now become quite fashionable in high-energy physics. Following the steps of MiniBooNE for analysis and particle identification and D0 for the first evidence and observation of single top quark production, other experiments and analyses are now using them routinely, in particular at the LHC.

Boosted decision trees are still a very active field of development, with academic groups and private companies testing their limits, providing new software [47, 49, 51] and using them to target recent issues like resistance to adversarial attacks (see e.g. [52, 53]).

References

- [1] B. P. Roe *et al.*, Boosted decision trees as an alternative to artificial neural networks for particle identification, *Nucl. Instr. Meth. A* **543** (2005) 577.
- [2] H.-J. Yang, B. P. Roe and J. Zhu, Studies of boosted decision trees for MiniBooNE particle identification, *Nucl. Instr. Meth. A* **555** (2005) 370.
- [3] D0 Collaboration, Evidence for production of single top quarks and first direct measurement of $|V_{tb}|$, *Phys. Rev. Lett.* **98** (2007) 181802.
- [4] D0 Collaboration, Evidence for production of single top quarks, *Phys. Rev. D* **78** (2008) 012005.
- [5] S. Ben-David *et al.*, A theory of learning from different domains, *Mach. Learn.* **79** (2009) 151.

- [6] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees* (Wadsworth, 1984).
- [7] ATLAS Collaboration, ATLAS b -jet identification performance and efficiency measurement with $t\bar{t}$ events in pp collisions at $\sqrt{s} = 13$ TeV, *Eur. Phys. J. C* **79** (2019) 970; arXiv:1907.05120 [hep-ex].
- [8] ATLAS Collaboration, Evidence for $t\bar{t}t\bar{t}$ production in the multilepton final state in proton–proton collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector, *Eur. Phys. J. C* **80** (2020) 1085; arXiv:2007.14858 [hep-ex].
- [9] G. Cowan *et al.*, Asymptotic formulae for likelihood-based tests of new physics, *Eur. Phys. J. C* **71** (2011) 1554; arXiv:1007.1727 [physics.data-an]. Erratum: *Eur. Phys. J. C* **73** (2013) 2501.
- [10] L. Moneta *et al.*, The RooStats project, *PoS ACAT2010* (2011) 057; arXiv:1009.1003 [physics.data-an].
- [11] C. Adam-Bourdarios *et al.*, The Higgs boson machine learning challenge, in *Proc. NIPS 2014 Workshop on High-energy Physics and Machine Learning* (2015).
- [12] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd edn., Springer Series in Statistics (Springer, 2009). <https://web.stanford.edu/~hastie/ElemStatLearn/>.
- [13] T. Hastie, R. Tibshirani and M. Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations*, Monographs on Statistics and Applied Probability (Chapman & Hall/CRC, 2015). <https://web.stanford.edu/~hastie/StatLearnSparsity/>.
- [14] N. Srivastava *et al.*, Dropout: A simple way to prevent neural networks from overfitting, *J. Mach. Learn. Res.* **15** (2014) 1929–1958.
- [15] A. J. Wyner, M. Olson, J. Bleich and D. Mease, Explaining the success of AdaBoost and random forests as interpolating classifiers, *J. Mach. Learn. Res.* **18** (2017) 1.
- [16] M. Belkin, D. Hsu, S. Ma and S. Mandal, Reconciling modern machine-learning practice and the classical bias–variance trade-off, *Proc. Natl. Acad. Sci. USA* **116** (2019) 15849; arXiv:1812.11118 [stat.ML].
- [17] I. Kononenko, Machine learning for medical diagnosis: history, state of the art and perspective, *Artif. Intell. Med.* **23** (2001) 89.
- [18] V. Podgorelec, P. Kokol, B. Stiglic and I. Rozman, Decision trees: An overview and their use in medicine, *J. Med. Syst.* **26** (2002) 445.
- [19] C. Gini, Variabilità e mutabilità, in *Memorie di Metodologica Statistica*, eds. E. Pizetti and T. Salvemini (Libreria Eredi Virgilio Veschi, Rome, 1955).
- [20] L. Breiman, Random forests, *Mach. Learn.* **45** (2001) 5.
- [21] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms* (Cambridge University Press, 2014). <https://www.cs.huji.ac.il/~shaish/UnderstandingMachineLearning>.
- [22] J. R. Quinlan, Simplifying decision trees, *Int. J. Man-Mach. Stud.* **27** (1987) 221.
- [23] J. H. Friedman and B. E. Popescu, Predictive learning via rule ensembles, *Ann. Appl. Stat.* **2** (2008) 916; arXiv:0811.1679 [stat.AP].

- [24] T. G. Dietterich, Machine learning research: Four current directions, *AI Magazine* **18** (1997) 97.
- [25] R. E. Schapire, The strength of weak learnability, *Mach. Learn.* **5** (1990) 197.
- [26] Y. Freund, Boosting a weak learning algorithm by majority, *Inf. Comput.* **121** (1995) 256.
- [27] Y. Freund and R. E. Schapire, Experiments with a new boosting algorithm, in *Proc. Thirteenth International Conf. Machine Learning, ICML'96* (1996).
- [28] Y. Freund and R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, *J. Comput. Syst. Sci.* **55** (1997) 119.
- [29] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms* (MIT Press, 2012).
- [30] J. H. Friedman, Greedy function approximation: A gradient boosting machine., *Ann. Statist.* **29** (2001) 1189.
- [31] J. H. Friedman, T. Hastie and R. Tibshirani, Additive logistic regression: A statistical view of boosting, *Ann. Statist.* **28** (2000) 337.
- [32] R. E. Schapire, Y. Freund, P. Bartlett and W. S. Lee, Boosting the margin: A new explanation for the effectiveness of voting methods, *Ann. Statist.* **26** (1998) 1651.
- [33] V. N. Vapnik, *The Nature of Statistical Learning Theory* (Springer, 2000).
- [34] L. Breiman, Arcing the edge, *Ann. Prob.* **26** (1998) 1683.
- [35] L. Breiman, Prediction games and arcing algorithms, *Neural Comput.* **11** (1999) 1493.
- [36] J. H. Friedman, Stochastic gradient boosting, *Comput. Stat. Data Anal.* **38** (2002) 367.
- [37] A. Hoecker *et al.*, TMVA — Toolkit for multivariate data analysis, preprint (2007); arXiv:physics/0703039 [physics.data-an].
- [38] Y. Freund, An adaptive version of the boost by majority algorithm, *Mach. Learn.* **43** (2001) 293–318.
- [39] ATLAS Collaboration, Measurement of the tau lepton reconstruction and identification performance in the ATLAS experiment using pp collisions at $\sqrt{s} = 13$ TeV, *ATLAS-CONF-2017-029* (2017).
- [40] T. Likhomanenko *et al.*, LHCb topological trigger reoptimization, *J. Phys. Conf. Ser.* **664** (2015) 082025; arXiv:1510.00572 [physics.ins-det].
- [41] L. Anderlini *et al.*, Muon identification for LHCb Run 3, preprint (2008); arXiv:2008.01579 [hep-ex].
- [42] CMS Collaboration, Observation of the diphoton decay of the Higgs boson and measurement of its properties, *Eur. Phys. J. C* **74** (2014) 3076; arXiv:1407.0558 [hep-ex].
- [43] ATLAS Collaboration, Search for the standard model Higgs boson produced in association with top quarks and decaying into a $b\bar{b}$ pair in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector, *Phys. Rev. D* **97** (2018) 072016; arXiv:1712.08895 [hep-ex].
- [44] ATLAS Collaboration, Identification of hadronic tau lepton decays using neural networks in the ATLAS experiment, *ATL-PHYS-PUB-2019-033* (2019).

- [45] ATLAS Collaboration, Deep sets based neural networks for impact parameter flavour tagging in ATLAS, *ATL-PHYS-PUB-2020-014* (2020).
- [46] L. Breiman, Bagging predictors, *Mach. Learn.* **24** (1996) 123.
- [47] T. Chen and C. Guestrin, XGBoost: A scalable tree boosting system, in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, KDD'16*. (Association for Computing Machinery, 2016); arXiv:1603.02754 [cs.LG].
- [48] ATLAS Collaboration, Observation of Higgs boson production in association with a top quark pair at the LHC with the ATLAS detector, *Phys. Lett. B* **784** (2018) 173; arXiv:1806.00425 [hep-ex].
- [49] G. Ke *et al.*, LightGBM: A highly efficient gradient boosting decision tree, in *Advances in Neural Information Processing Systems* (2017).
- [50] F. Pedregosa *et al.*, Scikit-learn: Machine learning in Python, *J. Mach. Learn. Res.* **12** (2011) 2825.
- [51] A. V. Dorogush, V. Ershov and A. Gulin, CatBoost: gradient boosting with categorical features support, preprint (2018); arXiv:1810.11363 [cs.LG].
- [52] H. Chen *et al.*, Robustness verification of tree-based models, in *Advances in Neural Information Processing Systems*, Vol. 32 (2019).
- [53] M. Andriushchenko and M. Hein, Provably robust boosted decision stumps and trees against adversarial attacks, in *Advances in Neural Information Processing Systems*, Vol. 32 (2019).

Chapter 3

Deep Learning from Four Vectors

Pierre Baldi*, Peter Sadowski† and Daniel Whiteson‡

**Department of Computer Science,
University of California Irvine
Irvine, California, USA
pfbaldi@uci.edu*

*†Department of Information and Computer Sciences,
University of Hawai'i at Mānoa
Honolulu, Hawai'i, USA
peter.sadowski@hawaii.edu*

*‡Department of Physics and Astronomy,
University of California Irvine
Irvine, California, USA
daniel@uci.edu*

1. Introduction: Pre-Deep Learning State-of-the-Art

The goal of the statistical analysis of particle physics data is to infer bounds on parameters of physical theories, such as the masses of particles or their rate of production in specific interactions. These statistical tasks, which involve classification, hypothesis testing, regression, and goodness-of-fit testing, require a statistical likelihood model $p(x|\theta)$ which describes the probability of observing experimental data x for specific values of the parameters θ of a physical theory.

Unfortunately, the statistical likelihood model can almost never be expressed analytically, due to the complex nature of the relationship between the theoretical parameters θ and the high-dimensional (10^2 – 10^8) data x . Instead, statistical models are typically estimated from samples generated using Monte Carlo methods [1, 2], whose computational expense limits the dimensionality of the feature space to $\mathcal{O}(10^0)$. In this context, it becomes vital to reduce the dimensionality of the data, and many initial applications [3–5] of machine learning to particle physics focused on development of classifiers, which offered powerful ways to perform this dimensional reduction and produce a single feature which summarizes much of the available information relevant to the statistical question.

Early applications of machine learning in physics [6, 7] were largely limited to shallow machine learning methods including boosted decision trees and artificial neural networks with a single hidden layer. This was primarily for historical reasons, including broad unawareness of the power of deep neural networks, misguided thinking and publications about local minima or vanishing gradients, as well as concerns over the problem of interpreting their output. In addition, it was well known that neural networks with a single hidden layer have universal approximation properties [8], although highly nonlinear functions may require an intractable number of hidden nodes. The combination of these circumstances led physicists to focus on shallow rather than deep machine learning methods, and shallow classifiers rapidly proliferated in physics (see [6] for an early application of neural networks). The approach successfully boosted the performance of many statistical analyses by allowing physicists to employ multiple observables, and was commonly referred to as “multi-variate analysis”. Such applications reduced the dimensionality of the feature space to one or two, allowing for estimation of the statistical models needed for inference tasks.

While dimensional reduction nearly always involves some loss of information, it does not necessarily reduce relevant information, as the statistical task usually only requires a subset of the information. The Neyman–Pearson lemma shows that the optimal decision boundary for a hypothesis test between two statistical models (in *any*

dimension feature space) can be determined by knowledge of their ratios; the full models are not needed. However, it was long suspected that shallow networks fell short of capturing all of the relevant information contained in the feature space. While it is not possible to directly estimate the absolute optimal performance without building an optimal classifier, it is possible to demonstrate that a given network is not optimal by finding a more powerful example.

A common experience in pre-deep-learning particle physics was to perform exhaustive feature engineering to simplify the task for the shallow classifier. A shallow network on four vectors,^a for example, would be compared to a shallow network with features built using domain knowledge. Such domain-specific expert features are generally nonlinear functions of four vectors that capture physical insights about the data. Almost invariably, the expert features would boost the performance, despite adding no unique information, demonstrating that the shallow classifiers had failed to discover these non-linear strategies on their own.

This feature-search approach is labor-intensive and not necessarily optimal; a robust machine learning method would obviate the need for this additional step and capture all of the available classification power directly from the raw data. Thus, the stage was set for deep learning.

2. Application of Deep Learning to Four Vectors

In this section, we describe a benchmark classification task [9, 10] that exemplifies a common use case for machine learning in particle physics: discrimination between signal and background processes. This example demonstrates a common failure mode of shallow networks on four vectors, which exhibit reduced performance compared to networks that use features engineered with domain knowledge.

^aFour vectors in momentum space are a generalization of three-dimensional momentum, including the total energy E , as (E, \vec{p}) and are typically used in particle physics to specify a particle's momentum and total energy.

2.1. Benchmark case for Higgs bosons

A typical classification task in particle physics distinguishes between a signal process, where new particles are produced, and one or more background processes, which mimic the nature and number of particles observed, but can be distinguished by their kinematics. An example [9] examined by experiments at the LHC is the production of a heavy electrically-neutral Higgs boson ($gg \rightarrow H^0$), which decays to a heavy electrically-charged Higgs boson (H^\pm) and a W boson [11, 12]. The H^\pm boson subsequently decays to a second W boson and the light Higgs boson, h^0 observed by the ATLAS [13] and CMS [14] experiments. The light Higgs boson decays predominantly to a pair of bottom quarks, giving the process:

$$gg \rightarrow H^0 \rightarrow W^\mp H^\pm \rightarrow W^\mp W^\pm h^0 \rightarrow W^\mp W^\pm b\bar{b}, \quad (1)$$

which leads to $W^\mp W^\pm b\bar{b}$ shown in Fig. 1. For the benchmark case here, $m_{H^0} = 425$ GeV and $m_{H^\pm} = 325$ GeV have been assumed.

The background process mimics this signal but without the Higgs boson intermediate state. It produces a pair of top quarks, each of which decay to Wb , also giving $W^\mp W^\pm b\bar{b}$.

Both processes yield the same set of observed particles: one charged lepton, four jets (two of which have b -tags) and missing transverse momentum. Together, the twenty-one individual momentum components of these particles comprise our *low-level feature set*.

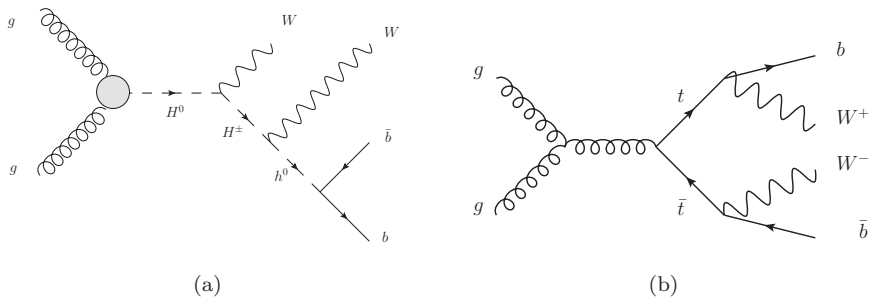


Fig. 1. Diagrams describing (a) the signal process involving new exotic Higgs bosons H^0 and H^\pm ; and (b) the background process involving top-quarks (t). In both cases, the resulting particles are two W bosons and two b -quarks.

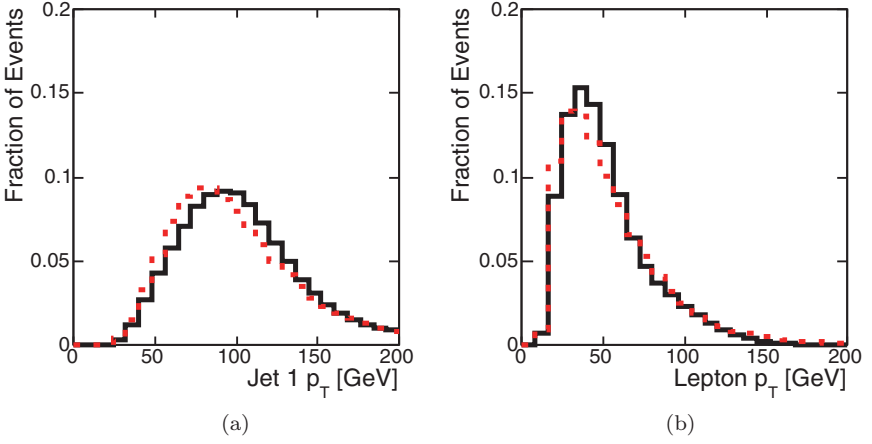


Fig. 2. Distributions of example low-level input features for Higgs benchmark. Shown are the simulated signal (black solid) and background (red dotted) marginal distributions of transverse momenta (p_T) of (a) the most energetic jet and (b) the lepton.

The low-level features show some differences between the signal and background processes — Fig. 2 shows the marginal distributions for two of these kinematic features. However, we see even larger differences in higher-level features constructed using domain knowledge of the different intermediate states. As the difference in the two hypotheses lies mostly in the existence of new intermediate Higgs boson states, it is possible to distinguish between the two hypotheses by attempting to identify whether the intermediate state existed by reconstructing its characteristic invariant mass. In the signal hypothesis we expect peaks in $m_{\ell\nu}$, m_{jj} , $m_{b\bar{b}}$, $m_{Wb\bar{b}}$, $m_{WWb\bar{b}}$, while the background should peak in $m_{j\ell\nu}$ and m_{jjj} . Figure 3 shows the difference in distributions for two of these high-level variables.

2.2. Performance

Deep neural networks (DNN) were compared to shallow neural networks (NN) and boosted decision trees (BDT) on three different subsets of input features: the low-level features only, the high-level features only, and both. Performance on the test set was measured

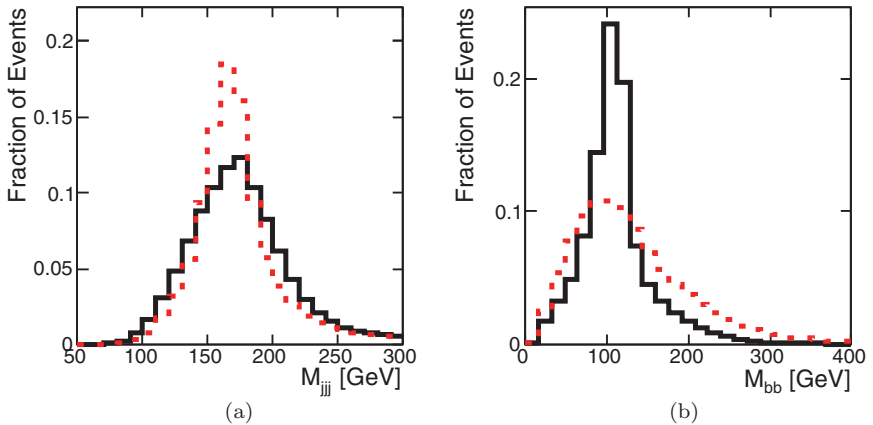


Fig. 3. Distributions of example high-level features for Higgs benchmark. Shown are the simulated signal (black) and background (red) events for marginal distributions of (a) the three-jet invariant mass and (b) the bottom-quark and anti-bottom-quark pair.

in terms of Area Under the ROC curve (AUC) and discovery significance (Table 1), as well as signal efficiency and background rejection; see Fig. 4.

The shallow neural networks and BDTs trained with the high-level features perform significantly better than those trained on only the low-level features, demonstrating the importance of feature engineering in shallow machine learning models. However, training all three methods with *only* the high-level features leads to lower performance than training with the complete set of features, indicating that the low-level features contain additional information that is not being captured by these engineered features. Only the deep learning approach shows nearly equal performance using the low-level features and the complete features. This suggests that it is automatically discovering high-level abstractions similar to those captured by the hand-engineered features, obviating the need for laborious feature engineering.

2.3. Discussion

It is widely accepted in experimental high-energy physics that machine learning is a powerful approach boosting statistical power

Table 1. Comparison of the performance of several learning techniques: boosted decision trees (BDT), shallow neural networks (NN), and deep neural networks (DNN) for three sets of input features: low-level features, high-level features and the complete set of features. Each neural network was trained five times with different random initializations. The table displays the mean Area Under ROC Curve (AUC) of the signal-rejection curve in Fig. 4, with statistical uncertainty measured in cross-validation shown in parentheses. Below is shown the expected significance of a discovery (in units of Gaussian σ) for 100 signal events and 1000 ± 50 background events.

Technique	Low-level	AUC	
		High-level	Complete
BDT	0.73 (0.01)	0.78 (0.01)	0.81 (0.01)
NN	0.733 (0.007)	0.777 (0.001)	0.816 (0.004)
DNN	0.880 (0.001)	0.800 (< 0.001)	0.885 (0.002)

Technique	Low-level	Discovery significance	
		High-level	Complete
NN	2.5σ	3.1σ	3.7σ
DNN	4.9σ	3.6σ	5.0σ

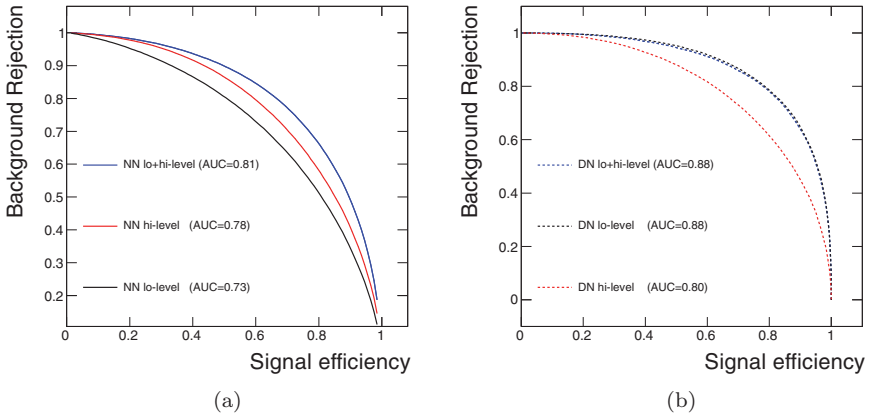


Fig. 4. Background rejection vs. signal efficiency on the Higgs benchmark for (a) shallow neural networks (NN) and (b) deep neural networks (DN). Curves are plotted for models trained using the low-level features only (black), the high-level features only (red), and the complete set of features (blue).

in exotic particle searches. Until the advent of deep learning, physicists reluctantly accepted the limitations of the shallow machine learning classifiers — laboriously constructing nonlinear feature combinations to help guide shallow networks and BDTs. This benchmark study shows that advances in deep learning can lift these limitations by *automatically* discovering powerful feature combinations directly from low-level features. Similar conclusions were reached in other benchmark cases, such as in searches for $t\bar{t}h$ production [15].

3. Parameterized Networks

The deep learning approach described above solves a simple signal vs. background classification task for a hypothesized particle with a particular mass. But the mass of a hypothesized particle is generally unknown. In practice, a hypothesized particle will have a *range* of possible masses, each of which would produce a different type of signal in the data. The classification tasks at different masses are closely related, but distinct. Physicists need a way to evaluate this *class* of signal hypotheses against the null (background) hypothesis.

A naive way to address this problem is to perform a finite number of individual comparisons. For each of K possible mass values, a hypothesized particle with that mass is simulated using Monte Carlo to produce a data set, and a machine learning model is trained to discriminate between it and the background. Methods from statistics can be used to account for the problem of multiple-hypothesis testing (e.g. the Bonferroni correction [16]). However, this naive approach is too conservative when the data distributions from different mass values are related. In practice, the distribution is expected to vary smoothly with the continuous parameter of the particle — that is, one expects similar mass values to result in similar distributions of observation data. This suggests the use of machine learning to model the relationship between the mass parameter and the data distribution.

A machine learning solution to this problem is to train a *single* classifier to perform particle searches for an entire range of possible mass values [17, 18]. This is done by extending the list of input features to include one or more additional parameters that describe the larger scope of the problem such as a new particle’s mass. The

approach can be applied to any classification model; however, neural networks provide a smooth interpolation in this new parameter space, while tree-based classifiers may not. A single parameterized network can replace a set of individual networks trained for specific cases, as well as smoothly interpolate to cases where it has not been trained. In the case of a search for a hypothetical new particle, this greatly simplifies the task — by requiring only one network — as well as making the results more powerful — by allowing them to be interpolated between specific values. In addition, they may outperform isolated networks by generalizing from the full parameter-dependent dataset. In this section we describe the use of parameterized neural networks and provide a realistic example. For a real application, see [19–22].

3.1. *Parameterized network structure and training*

A standard neural network takes as input a vector of features, \bar{x} , and computes a function of these features, $f(\bar{x})$. Parameterized networks address the case where the task is part of a larger context, described by one or more parameters, $\bar{\theta}$, by computing a function of both inputs: $f(\bar{x}, \bar{\theta})$. Thus, a parameterized neural network makes different predictions for input \bar{x} for different contexts $\bar{\theta}$; see Fig. 5.

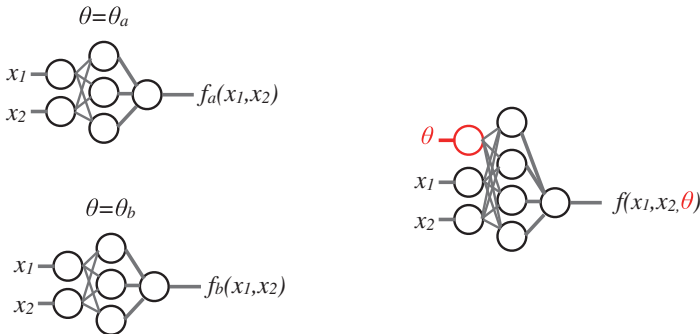


Fig. 5. Traditional neural networks (left) with input features (x_1, x_2) are trained with examples from fixed values of some latent parameter $\theta = \theta_a, \theta_b$. Neither network performs optimally for intermediate values of θ . A *parameterized* network (right) is trained with input features (x_1, x_2) as well as the input parameter θ ; such a network is trained with examples at several values of the parameter θ and interpolates for intermediate values. At test time, the user provides θ .

Unlike other networks, a parameterized network *requires* a value of $\bar{\theta}$ to perform inference on input \bar{x} , as the network output is a function of both.

Parameterized neural networks require some additional considerations during training. Each training example for such a parameterized network has the form $(\bar{x}, \bar{\theta}, y)_i$, where y is the target output. However, in classification problems $\bar{\theta}$ may not be meaningful for a particular target class. For example, the mass of a new particle is not meaningful for the background training examples. To avoid divulging information about y , one must randomly assign values [18] to $\bar{\theta}$ according to the same distribution used for the signal class.

Another issue is that the distribution of $\bar{\theta}$ in the training set represents a prior distribution that influences the final model, and should be specified carefully. Traditionally, $\bar{\theta}$ is determined by the hypothesis and fixed during detector simulations, producing samples from a conditional distribution $p(\bar{x}|\bar{\theta}, y)$; in parameterized neural networks, the training data would typically be generated by first sampling $\bar{\theta}$ from a class-specific prior $p(\bar{\theta}|y)$ then \bar{x} from $p(\bar{x}|\bar{\theta}, y)$. The robustness of the resulting parameterized classifier to the distribution of $\bar{\theta}$ in the training sample will depend on the physics encoded in the distributions $p(\bar{x}|\bar{\theta}, y)$ and how much they change with $\bar{\theta}$. The prior $p(\bar{\theta}|y)$ should be chosen carefully and should be considered when interpreting results, just as one would carefully consider a fixed $\bar{\theta}$ when building and evaluating a traditional classifier. In the studies presented below, the training data consists of equal sized samples for a few discrete values of $\bar{\theta}$ — the conditional distribution $p(\bar{x}|\bar{\theta}, y)$ varies smoothly enough in $\bar{\theta}$ that this reasonably approximates a uniform prior over $\bar{\theta}$.

3.2. *Physical example*

Parameterized networks address a common problem in searches for new particles of unknown mass, and we provide an illustrative example from [18]. Consider the search for a new particle X which decays to $t\bar{t}$, examining the most powerful decay mode in which $t\bar{t} \rightarrow W^+bW^-\bar{b} \rightarrow qq'b\ell\nu\bar{b}$. The dominant background is standard

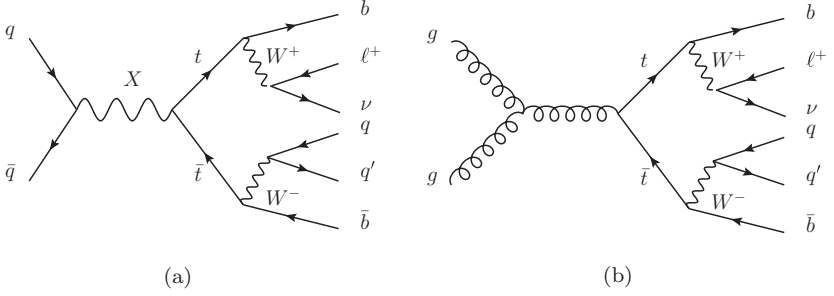


Fig. 6. Feynman diagrams showing (a) the production and decay of the hypothetical particle $X \rightarrow t\bar{t}$, as well as (b) the dominant standard model background process of top quark pair production. In both cases, the $t\bar{t}$ pair decay to a single charged lepton (ℓ), a neutrino (ν) and several quarks (q, b).

model $t\bar{t}$ production, which is identical in final state but distinct in kinematics due to the lack of an intermediate resonance. Figure 6 shows diagrams for the signal and background processes.

The set of event-level features include 21 low-level kinematic features resulting from reconstruction algorithms and 5 high-level features which incorporate physics domain knowledge. The distributions of four high-level features are shown in Fig. 7 to illustrate the differences between the signal distribution at different values of the particle mass, m_X , and the background distribution.

In order to test how well a parameterized neural network generalizes to new parameter values, an experiment compared the performance of a fixed neural network architecture trained on three different training data sets with different distributions of m_X , and tested on a data with $m_X = 1000$ GeV. The three different training data sets contained signal samples with different mass distributions: (1) $m_X = 1000$ GeV only; (2) $m_X = 500, 750, 1000, 1250, 1500$ GeV; and (3) $m_X = 500, 750, 1250, 1500$ GeV (no $m_X = 1000$ GeV). In each case, the training set contains 7M examples, the test set contains 1M, and approximately the same number of training and testing examples are used per mass point. On each data set, the same neural network architecture was trained, containing five 500-dimensional ReLU layers followed by a logistic output unit for binary classification. Parameters were initialized from a Gaussian distribution with mean zero

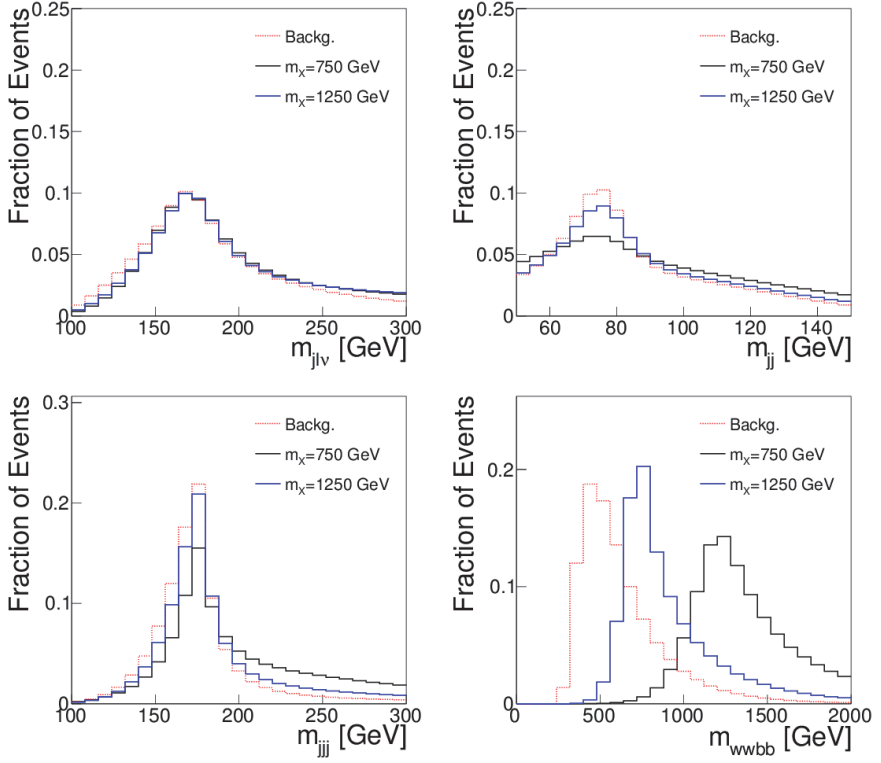


Fig. 7. Distributions [18] of high-level event features for the decay of $X \rightarrow t\bar{t}$ with two choices of m_X as well as the dominant background process; see text for definitions.

and width 0.1, and updated using stochastic gradient descent with mini-batches of size 100 and 0.5 momentum. The learning rate was initialized to 0.1 and decayed by a factor of 0.89 every epoch. Training was stopped after 200 epochs.

The results show that the parameterized network not only matches the performance of a network trained on a single mass value, but is able to generalize to mass values it has never seen before. Figure 8 shows that the parameterized network trained on $m_X = 500, 750, 1000, 1250, 1500$ GeV matches the performance of the fixed network trained on $m_X = 1000$ only. In the third data set, $m_X = 1000$ samples are removed from the training set so that the network must interpolate its solution, but the performance is

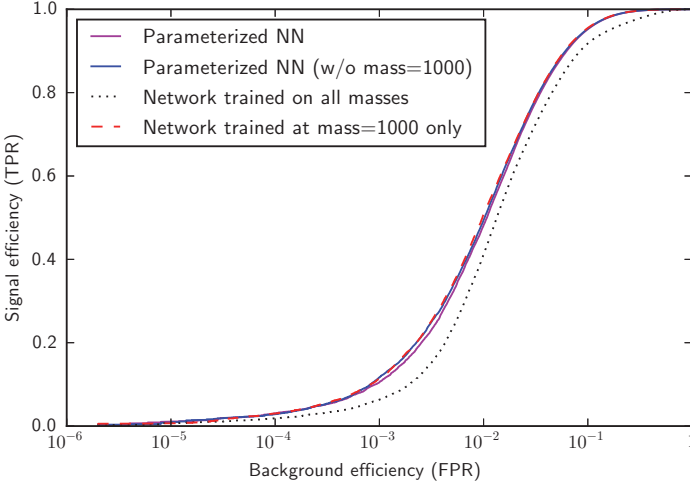


Fig. 8. Performance comparison [18] of signal-to-background discrimination for four classes of networks on a test sample with $m_X = 1000$ GeV. A parameterized network trained on all masses $m_X = 500, 750, 1000, 1250, 1500$ (magenta) performs just as well as a traditional network trained with only $m_X = 1000$ GeV (red). A second parameterized network trained with only $m_X = 500, 750, 1250, 1500$ is forced to interpolate the solution at $m_X = 1000$ GeV (blue), but performs equally well. However, a traditional non-parameterized network trained with all the mass points (black) shows a reduced performance. The results are indistinguishable for cases where the networks use only low-level features (shown) or low-level as well as high-level features (not shown, but identical).

unchanged, demonstrating that the parameterized network is able to generalize even in this high-dimensional example.

We note, however, that while the ability of the parameterized network was demonstrated in this case, and we expect this ability to generalize due to networks excellent performance in interpolation tasks, one cannot claim to predict similar quality of interpolation for an arbitrary task. Performance in a specific task would require a dedicated study.

The high dimensionality of this problem makes it difficult to visually explore the dependence of the neural network output on the parameter m_X . However, Fig. 9 compares the performance of the parameterized network to a single network trained at $m_X = 1000$ GeV when applied across the mass range of interest, a common

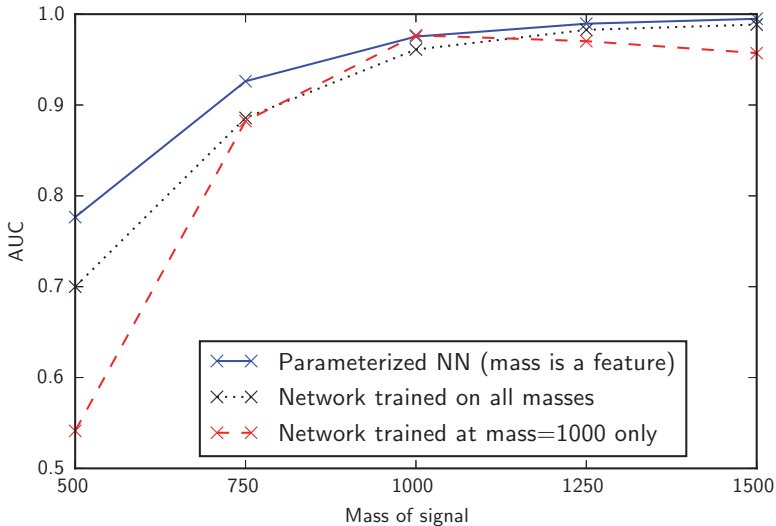


Fig. 9. Performance comparison [18] of signal-background discrimination for a network parameterized by mass (blue), a traditional network trained on all mass values (black), and a traditional network trained only on $m_X = 1000$ GeV. As expected, the network trained at a single mass shows decreasing AUC (from ROC curves in Fig. 8) as the mass deviates from the value in the training sample. The network trained on all masses does not perform optimally at $m_X = 1000$ GeV, but the parameterized network performs well at all mass values. The trend of improving AUC vs. mass reflects the increasing separation between the signal and background samples with mass, see Fig. 7.

use case. This demonstrates the loss of performance incurred by some traditional approaches and recovered in this approach. Similarly, we see that a single network trained an unlabeled mixture of signal samples from all masses has reduced performance at each mass value tested.

4. Handling Sets of Four Vectors

The deep neural network architectures discussed so far have consisted entirely of sequential layers, where each layer is fully-connected to the layer below. However, a key advantage of artificial neural network models is the ability to design neural network architectures that reflect properties of the data. These architecture design

choices enable us to constrain the class of functions to be considered, or more generally, incorporate an *implicit bias* for some functions over others. Examples include convolutional neural networks, Siamese neural networks, and various forms of recursive neural networks [23]. These architecture designs have been critical to the success of deep learning in computer vision, natural language processing, and bioinformatics. In this section, we examine the neural network design choices that can be used to handle *sets* of four vectors in physics.

Neural networks can be designed to have two key properties that are relevant to sets: invariance and equivariance. First, a function implemented by a neural network is *invariant* with respect to an operation if applying that operation to the input does not affect the output. A common example from deep learning is object detection with a convolutional neural network that is invariant to translations of the input image — the function output could be a single value corresponding to whether an object is present in the image, regardless of whether a translation operation is applied to the image. Second, a function implemented by a neural network is said to be *equivariant* with respect to an operation if applying that operation to the input results in a predictable change in the output. In a convolutional neural network, each convolutional *layer* is equivariant to translations because translating the input leads to a deterministic translation in the output representation.

For machine learning models that take sets as inputs, it is often desirable to have a model that is invariant or equivariant with respect to *permutation* of the set elements. For example, 3D vision models perform object detection from a set of 3D points on the surface of an object [24], and astronomy models predict the redshift of galaxies from a set of nearby galaxies [25, 26]. Similarly, exotic particle searches in physics involve classifying collision events based on sets of resulting four vectors. The function to be learned should be invariant to the ordering of the set elements. There are at least three ways to try to create supervised neural network models that are invariant to an operation: (1) data-augmentation, (2) canonicalization, and (3) architecture design. We discuss each in turn.

4.1. *Data-augmentation*

In data-augmentation, the training data is expanded by applying an operation to all training examples. In practice, it is usually more efficient to apply a random operation to the input data at training time. Either way, the network is forced to learn how to be invariant to that operation. For example, in object-detection models it is common to augment the data during training with random translations, rotations, and mirroring operations. On four vectors, ordering and boosting operations [27, 28] can augment the data. The disadvantage of this approach is that the model must *learn* that the output should be invariant — this requirement is not enforced by the model. Because this can make the learning problem much more difficult, it is typically used as a last resort when the other methods are not available.

4.2. *Canonicalization*

The second way to achieve invariance is through *canonicalization* of the input. In this method, the input is always mapped to some canonical element of the group defined by the operation, which enforces invariance without any other constraints on the model. An example is to enforce translational invariance in computer vision by “centering” an image at some deterministically-chosen point, or enforcing rotational invariance by rotating the image around that point until it is oriented along some canonical axis (as in [29] for jet substructure classification). For sets of four vectors, permutation invariance can be achieved by sorting the particles based on p_T , as is done in [30]. A potential disadvantage of this approach, besides having to come up with a good canonicalization scheme, is that the canonicalization procedure can introduce discontinuities in the function to be learned — a canonicalization that is sensitive to small changes in the inputs is undesirable, and could be worse than no canonicalization at all.

4.3. *Architecture design*

The third way to achieve equivariance and invariance in machine learning is through architecture design, where the hypothesis space

is constrained to functions that satisfy the condition. For example in convolutional neural network architectures, the convolution layers are equivariant to translations, and together with pooling layers they can be made invariant to translations. These architectures have been critical to the success of deep learning in computer vision [31]. Invariance to other input transformations can also be enforced through combinations of weight-sharing and pooling operations. Ideas from Lie group theory can be applied to this problem [32, 33]. One can trivially define neural network architectures that guarantee invariance to any transformation by defining an ensemble model that applies identical subnetworks to every possible transformation of the input, then pooling the result. Clearly this becomes intractable — or at least inefficient — for applications where the number of elements in the group is large or infinite, but there are often simpler approaches.

For input sets, permutation invariance can be achieved by: (1) applying an identical subnetwork to each set element, using shared weights; then (2) pooling the output. The shared weights result in equivariance to permutations of the inputs, since the new outputs will be equivalent to the permutation of the original outputs. The second step achieves invariance, e.g. with max or mean pooling of the possibly-multidimensional outputs. Designing neural network architectures that account for data symmetries like permutation invariance is one example of incorporating physics knowledge into the machine learning model, which is discussed more in the next section.

5. Physics-aware Networks

In applying machine learning to physics problems, one is often presented with the challenge of bringing physics knowledge to bear on the machine learning models [34, 35]. This situation can present itself in different forms: choosing of the relevant input and output variables, adding priors or regularization terms in the loss function, or imposing constraints on the neural architectures. Each of these contributes to explicit or implicit model *bias*, which can greatly affect the resulting performance. Often it is difficult to predict how these choices will affect performance, so they are treated

as hyperparameters and optimized by trying different variations. Here we consider two different situations corresponding to physics-informed architecture design and incorporation of physics constraints.

5.1. *Physics-informed architecture design*

The permutation-invariant models described above are one example of incorporating domain-knowledge into a neural network architecture. We can design neural network architectures that account for additional physics knowledge by taking advantage of other architecture design motifs. These include the local connectivity, weight sharing, and pooling of convolutional neural networks, but also skip connections [36], gating [37, 38], and attention [39–41]. We briefly discuss two other physics-informed neural network architectures applicable to four vectors.

One example of a physics-informed neural network architecture is [34]. Decaying particles in the detector typically result in decay products that are hierarchically clustered in space and time (jet substructures). Thus, sets of four vectors often have additional structure that can be exploited. When the clustering hierarchy of each event can be reconstructed, for example using a sequential recombination jet algorithm [42], this additional information can be incorporated into the network. *Recursive* neural network architectures can be constructed to match the topology of the jet clustering algorithms, analogous to models from Natural Language Processing that take advantage of sentence parse trees [43, 44]. The recursive physics neural network architecture is constructed on a *per-event* basis to reflect the tree structure of that event. In addition to the properties of permutation invariance (assuming each node is permutation invariant) and scalability to an arbitrary number of set elements, this model has the additional property of *local connectivity* among related elements in the set, which can lead to better generalization.

Another example is the Lorentz-Boosted Neural Networks in [45], in which the first hidden layer of the network is interpreted as “composite particles” and corresponding “rest frames,” and represented as

linear combinations of the input four vectors. Each learned composite particle is then boosted into its corresponding rest frame using the nonlinear Lorentz transformation. The resulting feature representations are then fed into a neural network, and the entire system is trained using back-propagation. The major advantage of this architecture is that it constrains the representation of the data into a form that is readily interpreted by physicists (i.e. Lorentz-transformed four vectors) and for which physically meaningful features can be extracted such as invariant masses, pseudorapidities, and so forth.

Yet another example is the approach described in [46], which uses recursive neural networks, of the form of transformer architectures [41, 47] used in language processing and tensor attention mechanisms, applied to many-jet event reconstruction in a manner that is invariant to any permutation of the four vectors in the variable-size input set.

5.2. *Incorporating physics constraints*

Here, we consider the situation where there are physical laws, in the form of exact equations, relating the values of some of the relevant variables. In addition to physics, many fields of science and engineering (e.g. fluid dynamics, hydrology, solid mechanics, chemistry kinetics) have exact, often *analytic*, closed-form constraints, i.e. constraints that can be explicitly written using analytic functions of the system’s variables. Examples include translational or rotational invariance, conservation laws, or equations of state. While physically-consistent models should enforce constraints to within machine precision, data-driven algorithms often fail to satisfy well-known constraints that are not explicitly enforced. In particular, while neural networks may provide powerful classification and regression tools for nonlinear systems, they may optimize overall performance while violating these constraints on individual samples.

Despite the need for physically-informed neural networks for complex physical systems [48–51], enforcing constraints [52] has been limited mostly to physical systems governed by specific equations, such as advection equations [53–55], Reynolds-averaged Navier–Stokes equations [56, 57], or quasi-geostrophic equations [58]. Thus, it is

necessary to have methods that can enforce analytic constraints in more general settings. Here, we describe two general ways for enforcing constraints, first in a soft way, and then in a hard way.

In general, let us assume that there is a constraint of the form $\mathcal{C}(x, y, z) = 0$ that must be satisfied by the input variables x , the output variables y , and possibly some auxiliary variables z . If \mathcal{E} is the error function of the neural network trained on the pairs (x, y) , we can enforce the constraints in a soft way by adding a penalty term to the loss function, e.g. using a new loss function of the form $\mathcal{E}' = \mathcal{E} + \lambda \mathcal{C}^2$ where λ is an additional hyperparameter controlling the strength of the corresponding regularization (or equivalently log prior) terms. This approach has been used for instance in climate modeling [59–61]. While this approach can be effective, there is no guarantee that the constraints may not be violated.

A general way for enforcing constraints in a hard way is described in [62]. There are several possible implementation of this idea, but the gist of it is to augment the basic neural architecture with an additional neural network to enforce the constraints. For this, we can first decompose y non-uniquely as $y = (y_1, y_2)$. Then, we introduce a first neural network with adaptive weights that produces an output y'_1 , trying to predict y_1 from x . This is followed by a second network which computes y'_2 from x, y'_1 and z , enforcing the constraint C to machine precision. The weights of the second network are fixed and determined by the knowledge of C . For instance, the second network can be linear if the constraint C is linear. We can then combine the two networks into a single overall architecture whose final output is the vector (y'_1, y'_2) . This output always satisfies the constraint C by construction. Furthermore, it can be compared to the target (y_1, y_2) and the resulting errors can be backpropagated through the combined network, through both the fixed and adjustable weights. As a result of this approach, the constraint C is satisfied at all times, both during and after learning.

6. Conclusions

We have reviewed the advent of deep learning in high-energy physics, first used in classification tasks operating on four-vector features

before being applied to tracks, images, graphs, and low-level detector data. Even in the case of four-vectors where the number of features is relatively small, deep learning can be used to improve classification performance and incorporate domain knowledge in various forms. In particular, neural networks can be designed to model a set of related functions using parameterized networks, capture permutation invariance in sets with weight-sharing and pooling, and incorporate additional physics constraints through architecture design or augmented loss functions.

References

- [1] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer, and T. Stelzer, MadGraph 5: Going beyond, *J. High Energy Phys.* **06** (2011) 128; arXiv:1106.0522 [hep-ph].
- [2] GEANT4, S. Agostinelli *et al.*, GEANT4—a simulation toolkit, *Nucl. Instrum. Meth. A* **506** (2003) 250.
- [3] H. Abramowicz, A. Caldwell, and R. Sinkus, Neural network based electron identification in the ZEUS calorimeter, *Nucl. Instrum. Meth. A* **365** (1995) 508; arXiv:hep-ex/9505004.
- [4] DØ Collaboration, Search for single top quark production at DØ using neural networks, *Phys. Lett. B* **517**(3–4) (2001) 282.
- [5] DELPHI Collaboration, Classification of the hadronic decays of the Z^0 into b and c quark pairs using a neural network, *Phys. Lett. B* **295**(3) (1992) 383.
- [6] R. Vazquez, F. Halzen and E. Zas, Improving the Čerenkov imaging technique with neural networks, *Phys. Rev. D* **45**(1) (1992) 356.
- [7] P. Baldi, *Deep Learning in Science: Theory, Algorithms, and Applications* (Cambridge University Press, Cambridge, 2021), in press.
- [8] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Netw.* **2**(5) (1989) 359.
- [9] P. Baldi, P. Sadowski and D. Whiteson, Searching for exotic particles in high-energy physics with deep learning, *Nature Commun.* **5** (2014) 4308; arXiv:1402.4735 [hep-ph].
- [10] P. Baldi, P. Sadowski, and D. Whiteson, Enhanced Higgs boson to $\tau^+\tau^-$ search with deep learning, *Phys. Rev. Lett.* **114**(11) (2015), 111801; arXiv:1410.3469 [hep-ph].
- [11] CDF Collaboration, Search for a two-Higgs-boson doublet using a simplified model in $p\bar{p}$ collisions at $\sqrt{s} = 1.96$ TeV, *Phys. Rev. Lett.* **110**(12) (2013) 121801; arXiv:1212.3837 [hep-ex].
- [12] ATLAS Collaboration, Search for a multi-Higgs-boson cascade in $W^+W^-b\bar{b}$ events with the ATLAS detector in pp collisions at $\sqrt{s} = 8$ TeV, *Phys. Rev. D* **89** (2014) 032002; arXiv:1312.1956 [hep-ex].

- [13] ATLAS Collaboration, Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC, *Phys. Lett. B* **716** (2012) 1; arXiv:1207.7214 [hep-ex].
- [14] CMS Collaboration, Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC, *Phys. Lett. B* **716** (2012) 30; arXiv:1207.7235 [hep-ex].
- [15] R. Santos, M. Nguyen, J. Webster, S. Ryu, J. Adelman, S. Chekanov and J. Zhou, Machine learning techniques in searches for $t\bar{t}h$ in the $h \rightarrow b\bar{b}$ decay channel, *JINST* **12**(04) (2017) P04014; arXiv:1610.03088 [hep-ex].
- [16] O. J. Dunn, Multiple Comparisons among Means, *J. Amer. Statist. Assoc.* **56**(293) (1961), 52.
- [17] K. Cranmer, J. Pavez, and G. Louppe, Approximating likelihood ratios with calibrated discriminative classifiers, preprint (2015); arXiv:1506.02169 [stat.AP].
- [18] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski and D. Whiteson, Parameterized neural networks for high-energy physics, *Eur. Phys. J. C* **76**(5) (2016) 1.
- [19] CMS Collaboration, Search for resonant and nonresonant higgs boson pair production, *J. High Energy Phys.* **2018**(1) (2018) 54; arXiv:1808.00336 [hep-ex].
- [20] ATLAS Collaboration, Performance of mass-decorrelated jet substructure observables for hadronic two-body decay tagging in ATLAS, preprint, *ATL-PHYS-PUB-2018-014* (2018); <https://cds.cern.ch/record/2630973>.
- [21] CMS Collaboration, A deep neural network to search for new long-lived particles decaying to jets, preprint (2019); arXiv:1912.12238 [hep-ex].
- [22] CMS Collaboration, Search for a charged Higgs boson decaying into top and bottom quarks in events with electrons or muons in proton-proton collisions at $\sqrt{s} = 13$ TeV, *J. High Energy Phys.* **01** (2020) 096; arXiv:1908.09206 [hep-ex].
- [23] P. Baldi, The inner and outer approaches to the design of recursive neural architectures, *Data Mining knowledge Discovery* **32**(1) (2018), 218.
- [24] C. R. Qi, H. Su, K. Mo and L. J. Guibas, PointNet: Deep learning on point sets for 3D classification and segmentation, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2017).
- [25] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov and A. J. Smola, *Deep sets*, in *Advances in Neural Information Processing Systems 30*, (2017) pp. 3391–3401.
- [26] R. Beck, P. Sadowski, Y. Glaser and I. Szapudi, Refined redshift regression in cosmology with graph convolution networks, in *NeurIPS Machine Learning and the Physical Sciences Workshop* (2019).
- [27] A. Butter, G. Kasieczka, T. Plehn and M. Russell, Deep-learned top tagging with a Lorentz layer, *SciPost Phys.* **5**(3) (2018) 028; arXiv:1707.08966 [hep-ph].
- [28] J. Pearkes, W. Fedorko, A. Lister and C. Gay, Jet constituents for deep neural network based top quark tagging, preprint (2017); arXiv:1704.02124 [hep-ex].

- [29] P. Baldi, K. Bauer, C. Eng, P. Sadowski, and D. Whiteson, Jet substructure classification in high-energy physics with deep neural networks, *Phys. Rev. D* **93** (2016) 094034.
- [30] D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban and D. Whiteson, Jet flavor classification in high-energy physics with deep neural networks, *Phys. Rev. D* **94**(11) (2016) 112002.
- [31] G. Urban, K. J. Geras, S. E. Kahou, O. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose and M. Richardson, Do deep convolutional nets really need to be deep and convolutional? preprint (2016); arXiv:1603.05691 [stat.ML]
- [32] T. Cohen and M. Welling, Group equivariant convolutional networks, in *Proc. 33rd Int. Conf. Machine Learning*. PMLR, New York, New York, USA, 20–22 June, (2016).
- [33] T. Cohen, M. Weiler, B. Kicanaoglu and M. Welling, Gauge equivariant convolutional networks and the icosahedral CNN, in *Proc. 36th Int. Conf. Machine Learning*. PMLR, Long Beach, California, USA, 09–15 June (2019).
- [34] G. Louppe, K. Cho, C. Becot and K. Cranmer, QCD-Aware recursive neural networks for jet physics, *J. High Energy Phys.* **01** (2019) 057; arXiv:1702.00748 [hep-ph].
- [35] T. Cheng, Recursive neural networks in quark/gluon tagging, *Comput. Softw. Big Sci.* **2**(1); (2018) 3; arXiv:1711.02633 [hep-ph].
- [36] O. Ronneberger, P. Fischer, and T. Brox, U-net: Convolutional networks for biomedical image segmentation, in *Int. Conf. Medical Image Computing and Computer-Assisted Intervention* (Springer, 2015).
- [37] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Computation* **9**(8) (1997) 1735.
- [38] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in *Proc. 2014 Conf. Empirical Methods in Natural Language Processing (EMNLP)*. (2014).
- [39] D. Bahdanau, K. Cho and Y. Bengio, Neural machine translation by jointly learning to align and translate, *ICLR* (2014); arXiv:1409.0473 [cs.CL].
- [40] T. Luong, H. Pham and C. D. Manning, Effective approaches to attention-based neural machine translation, in *Proc. 2015 Conf. Empirical Methods in Natural Language Processing* (2015).
- [41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, Attention is all you need, in *Advances in Neural Information Processing Systems* (2017).
- [42] M. Cacciari, G. P. Salam and G. Soyez, The anti- k_t jet clustering algorithm, *J. High Energy Phys.* **04** (2008) 063; arXiv:0802.1189 [hep-ph].
- [43] C. Goller and A. Kuchler, Learning task-dependent distributed representations by backpropagation through structure, in *IEEE Int. Conf. Neural Networks, 1996* (IEEE, 1996).

- [44] R. Socher, C. C. Lin, C. D. Manning and A. Y. Ng, Parsing natural scenes and natural language with recursive neural networks, in *Proc. 28th Int. Conf. Machine Learning (ICML-11)* (2011).
- [45] M. Erdmann, E. Geiser, Y. Rath and M. Rieger, Lorentz boost networks: autonomous physics-inspired feature engineering, *JINST* **14**(06) (2019) P06006.
- [46] M. Fenton, A. Shmakov, T. Ho, S. Hsu, D. Whiteson and P. Baldi, Permutationless many-jet event reconstruction with symmetry preserving attention networks, preprint (2020); arXiv:2010.09206.
- [47] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, preprint (2018); arXiv:1810.04805.
- [48] M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais and Prabhat, Deep learning and process understanding for data-driven Earth system science, *Nature* **566**(7743) (2019) 195.
- [49] K. J. Bergen, P. A. Johnson, M. V. de Hoop and G. C. Beroza, Machine learning for data-driven discovery in solid earth geoscience, *Science* **363** (2019) 6433.
- [50] A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova and V. Kumar, Theory-guided data science: A new paradigm for scientific discovery from data, *IEEE Trans. Knowledge Data Eng.* **29**(10) (2017) 2318.
- [51] J. Willard, X. Jia, S. Xu, M. Steinbach and V. Kumar, Integrating physics-based modeling with machine learning: A survey, preprint (2020); arXiv:2003.04919.
- [52] P. Márquez-Neila, M. Salzmann and P. Fua, Imposing hard constraints on deep networks: Promises and limitations, preprint (2017); arXiv:1706.02025.
- [53] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations, preprint (2017); arXiv:1711.10561.
- [54] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, Learning data-driven discretizations for partial differential equations, *Proc. Natl. Acad. Sci.* **116**(31) (2019) 15344.
- [55] E. de Bezenac, A. Pajot, and P. Gallinari, Deep learning for physical processes: Incorporating prior scientific knowledge, *J. Statist. Mech. Theory Exp.* **2019**(12) (2019) 124009.
- [56] J. Ling, A. Kurzawski, and J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, *J. Fluid Mech.* **807** (2016) 155.
- [57] J. L. Wu, H. Xiao and E. Paterson, Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework, *Phys. Rev. Fluids* **3**(7) (2018) 074602.
- [58] T. Bolton and L. Zanna, Applications of deep learning to ocean data inference and subgrid parameterization, *J. Adv. Model. Earth Syst.* **11**(1) (2019) 376.

- [59] A. Karpatne, W. Watkins, J. Read and V. Kumar, Physics-guided neural networks (PGNN): An application in lake temperature modeling, preprint (2017); arXiv:1710.11431.
- [60] X. Jia, J. Willard, A. Karpatne, J. Read, J. Zwart, M. Steinbach, and V. Kumar, Physics guided RNNs for modeling dynamical systems: A case study in simulating lake temperature profiles, in *SIAM Int. Conf. Data Mining, SDM 2019* (2019); arXiv:1810.13075.
- [61] M. Raissi, A. Yazdani, and G. E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, *Science* **367**(6481) (2020) 1026.
- [62] T. Beucler, M. Pritchard, S. Rasp, J. Ott, P. Baldi and P. Gentine, Enforcing analytic constraints in neural-networks emulating physical systems, preprint (2019); arXiv:1909.00912 [physics.comp-ph].

This page intentionally left blank

Chapter 4

Anomaly Detection for Physics Analysis and Less Than Supervised Learning

Benjamin Nachman

*Lawrence Berkeley National Laboratory,
Berkeley, CA 94720, USA
bpnachman@lbl.gov*

Modern machine learning tools offer exciting possibilities to qualitatively change the paradigm for new particle searches. In particular, new methods can broaden the search program by gaining sensitivity to unforeseen scenarios by learning directly from data. There has been a significant growth in new ideas and they are starting to be applied to experimental data. This chapter introduces these new anomaly detection methods, which range from fully supervised algorithms to unsupervised, and include weakly supervised methods.

1. Introduction

Searching for new particles and forces of nature is one of the main goals of HEP. Despite hints for new fundamental structure, there has been no convincing evidence since the discovery of the Higgs Boson in 2012 [1, 2]. All of the major HEP experiments are engaged in an extensive search program and the goal of this chapter is to explore the dependence of these efforts on particular models and to examine how machine learning may be able to significantly broaden existing efforts.

In particular, the main topic of this chapter is machine learning-based *anomaly detection*. An anomaly is an unexpected feature of the data and as such all searches for new particles are anomaly detection analyses. The main feature of the searches described in this chapter that separates them from others is the level of model dependence in

various parts of the analysis. In particular, this chapter will focus on searches that rely as little as possible on signal and background models for both achieving sensitivity to new physics as well as calibrating the background. This is in contrast to most dedicated searches that are optimized with a particular signal model. All of these concepts will be made more precise below.

This chapter is organized as follows. Section 2 introduces traditional searches for new particles in HEP and how model dependence plays a role in the analysis development and statistical procedure. The remaining sections review various types of machine learning-based anomaly detection approaches. These sections are organized by how labeled examples are (or are not) provided to train machine learning classifiers (*supervision*). In particular, the chapter covers supervised learning (Secs. 3 and 4), unsupervised learning (Sec. 5), weakly supervised learning (Sec. 6), and lastly hybrid approaches (Sec. 7). Recent results from ATLAS and CMS are highlighted in Sec. 8 and the chapter ends with conclusions and outlook in Sec. 9.

2. Model Dependence in HEP Data Analysis

A typical search for new phenomena begins by selecting a target signal model or class of models, S . Then, simulations of the signal and of the Standard Model (SM) background B are performed. These simulations are used to inform the construction of a test statistic λ that is based on features $x \in \mathbb{R}^N$. A threshold c is chosen so that $\Pr(\lambda > c|S + B)$ is significantly larger than $\Pr(\lambda > c|B)$. A combination of simulations and data-driven methods are used to estimate these tail probabilities given the observed data. A discovery is declared when the data are much more consistent with the $S + B$ hypothesis than the B -only hypothesis.^a If instead the data are more consistent with the B -only hypothesis, then one can set limits on the production cross section of a hypothetical signal.

^aIt could also be that the data are inconsistent with both hypotheses. This is one of the motivations for the widely used CL_s procedure [3].

In this paradigm, models are used in two important ways. First, models of both S and B are used to choose λ . In the absence of nuisance parameters and for a single signal model, the Neyman–Pearson lemma [4] states that for a fixed probability of rejecting the null hypothesis when it is true (level or type I error rate), the probability for rejecting the null hypothesis when the alternative is true (power) is maximized with the likelihood ratio test statistic. Typically, λ is chosen manually by scanning features that are known to enhance the likelihood ratio. A growing number of searches use various machine learning methods to strive for optimal performance. Machine learning methods can also be used to extend beyond a “cut-and-count” approach to achieve an unbinned optimal test statistic [5]. When a search involves profiling nuisance parameters or composite alternative hypotheses (multiple signal models), there is no uniformly most powerful test statistic. In these cases, the likelihood ratio is still a reasonable target and there are also a variety of inference-aware methods for achieving optimality [6, 7].

Second, models are used to determine the p -values $\Pr(\lambda > c|S+B)$ and $\Pr(\lambda > c|B)$. In some cases, simulations are used to directly estimate these probabilities using Monte Carlo (MC) methods. In many cases, data are used as part of the density estimation. The extent to which data can be used depends on assumptions about the background and the signal in the targeted region of phase space. Even if the p -values are ultimately computed entirely from data using parametric or non-parametric methods, simulations often play a key role in validating the assumptions that justify the data-based procedure or in deriving method non-closure uncertainties.

One can categorize a search strategy by its S - and B -dependence for both the signal sensitivity and background specificity, as shown in Fig. 1. As described above, most searches can be placed in the lower left part of Fig. 1(a) (signal sensitivity). Searches with a clear signal hypothesis that can be accurately simulated, but with a complex or hard-to-simulate background are found in the upper left part of Fig. 1(a). The searches depicted in the lower right corner of Fig. 1(a) will be discussed further in Sec. 3 and the remaining search strategies illustrated in the top right corner of Fig. 1(a) will be described

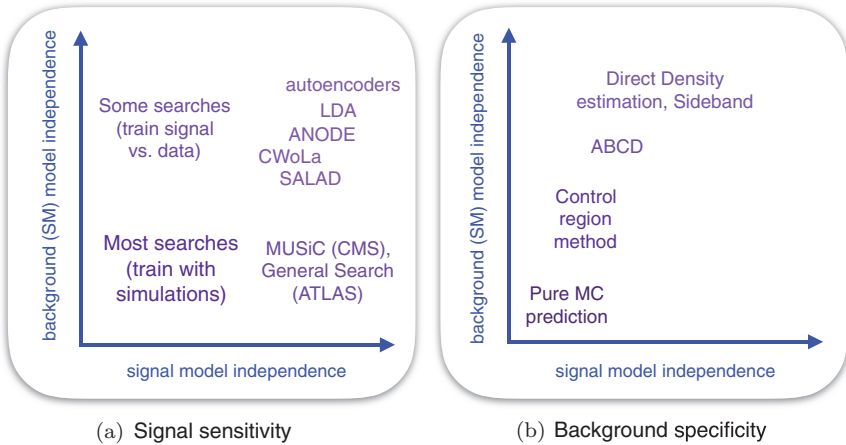


Fig. 1. A graphic depicting the classification of searches by their dependence on models of the signal and Standard Model (SM) background for both (a) gaining signal sensitivity (choosing the test statistic λ) and (b) calibrating the background p -values. The Model Unspecific Search for New Physics (MUSiC) [8–10] and General Search [11–13] are results from the CMS and ATLAS Collaborations, respectively. LDA stands for *Latent Dirichlet Allocation* [14, 15], ANODE stands for *ANOMaly detection with Density Estimation* [16], SALAD stands for *Simulation Assisted Likelihood-free Anomaly Detection* [17] and CWoLa stands for *Classification Without Labels* [18–20]. Direct density estimation is a form of side-banding where the multidimensional feature space density is learned conditional on the resonant feature. Figure reproduced from [16].

in Secs. 5, 6, and 7. The notion of optimality is more complicated for these *anomaly detection* approaches because there is no single signal model hypothesis. However, one can still consider optimality in the context of a different hypothesis test: data vs. background. Let $p_{\text{background}}(x|B)$ be the density describing the background and $p_{\text{data}}(x)$ be the density observed in a signal region. One can view anomaly detection as positing $p_{\text{background}}(x|B)$ as the null hypothesis with $p_{\text{data}}(x)$ as the alternative hypothesis. By construction, the data are consistent with the alternative hypothesis, but this does not mean that the null hypothesis can be rejected. As a simple hypothesis test, the Neyman–Pearson lemma guarantees that $p_{\text{background}}/p_{\text{data}}$ is an optimal test statistic. An anomaly detection technique is defined to be *asymptotically optimal* if there is some limit in which it approaches

this optimal test statistic. While many of the techniques in Secs. 6 and 7 are asymptotically optimal, most of the methods presented in Sec. 5 are not.

Achieving signal sensitivity is not sufficient to produce a physics result — one must also calibrate the background. Said another way, selecting anomalous events is irrelevant if there is no context to provide information on their strangeness. Precise background calibration is a key difference in anomaly detection between industry and HEP. In the former, anomalies are often off-manifold (e.g. a flying elephant) instead of local over-densities (e.g. more elephants than expected at a watering hole) and so it is less important that the background rate be known precisely. A variety of common methods are highlighted in Fig. 1(b). Pure MC estimation is reserved for final states that are relatively simple and precisely known theoretically and experimentally such as pure electroweak processes with only charged leptons [21]. Many searches use the *control region method* whereby simulations are calibrated using event selections that are close to the final phase space, but sufficiently far away that the expected signal purity is low. The requirement on signal purity introduces a dependence on the signal model. Two “fully data-driven” approaches are the ABCD and sideband methods. In both approaches, the data are used directly to transport predictions from a control region to the signal sensitive region instead of relying on simulation for this extrapolation. In the ABCD method, two classifiers f and g and two working points a and b are constructed and then four regions called A, B, C and D are defined by $f \leq a$ and $g \leq b$ (for a machine learning version of ABCD, see [22–24]). If f and g are independent, then one can relate the background prediction in the region $f > a$ and $g > b$ to the other three regions. The ABCD method depends on the background model by verifying independence and on the signal model by verifying low-signal contamination in the sidebands. The sideband method requires knowledge of one feature where the signal should be localized and the background is not localized. Regions away from the signal are then defined as sidebands and a fit can be used to interpolate the background prediction into the signal region. Resonant new physics is expected to be localized at the mass of the new particle, so this is

a moderately weak assumption. Relatively simple parametric models are often used for the fit, although machine learning approaches have also been proposed for this purpose [16, 17, 25, 26].

A core requirement of the sideband method is that the background is not localized where the potential signal should be localized. While this is generally true inclusively because background processes are often non-resonant, machine learning classifiers can artificially introduce localized features in the background. This often happens if a classifier is trained to distinguish a resonant signal from the SM background and a feature sensitive to the mass of the new particle is used in the training. One can simply remove mass-sensitive features from the training, but powerful classifiers can learn the mass indirectly through subtle correlations with other useful features. A variety of decorrelation techniques exist to solve this problem [22, 27–40]. In the context of neural networks, one can add terms to the loss function to achieve automatic decorrelation:

$$\mathcal{L}(f) = \sum_i \mathcal{L}_{\text{classifier}}(f(x_i), y_i) + \alpha (1 - y_i) \mathcal{L}_{\text{decorrelation}}(f(x_i), m_i), \quad (1)$$

where $\alpha \geq 0$ is a hyperparameter, $\mathcal{L}_{\text{classifier}}$ is the usual classifier loss such as binary cross entropy or mean squared error, f is the classifier, x are the features, y are the labels ($y = 0$ is background), and m is the feature that needs to be independent from f . In one approach, $\mathcal{L}_{\text{decorrelation}}$ is itself a neural network [31] that is designed to learn m from f . This adversarial method is optimized as a minimax solution whereby the second neural network is as bad as possible when decorrelation is achieved. Another possibility is for^b $\mathcal{L}_{\text{decorrelation}}$ to be a measure of dependence between $f(x)$ and m such as the distance correlation [34]. The later introduces only one free parameter (α) compared with thousands of parameters in adversarial method, but may be less flexible and can require large batches during training.

^bTechnically, for the distance correlation, this loss is applied at the level of a batch because it requires computing expectation values over pairs of events.

It is also possible to relax the decorrelation assumption by allowing for a controlled dependence on m [40].

In practice, analyses can mix and match strategies from Figs. 1(a) and 1(b). Some methods are naturally paired, such as those described in Secs. 6 and 7, as they were developed in the context of the side-band method. Decorrelation techniques that were first introduced for background estimation can also be important for signal sensitivity. The interplay between independence and signal sensitivity is discussed in Secs. 6 and 7. The remainder of this chapter focuses on machine learning methods for achieving signal sensitivity.

3. Signal Independent, Background Model Dependent

In some sense, all searches for physics beyond the Standard Model are anomaly detection analyses because anything discovered would be anomalous by definition. Furthermore, many searches publish “model-independent limits”. Such limits are signal model-independent only in the cross-section, but are strongly signal model-dependent in the acceptance. The rest of this chapter will instead focus on searches that do not target a particular signal model, although they often target a class of models such as resonance decays into a particular final state. Many searches are relatively inclusive and not optimized for a particular signal (e.g. the inclusive dijet search at the LHC [41, 42]). While such searches are broadly sensitive, this chapter will not focus on them because they are not actively optimized for sensitivity to new physics aside from general considerations (e.g. avoid large rapidity gaps in the case of dijets). Signal model-independent searches have a long history in HEP and have been performed by D0 [43–46], H1 [47, 48], ALEPH [49], CDF [50–52], CMS [8–10, 53] and ATLAS [11–13]. The general strategy in these analyses is to directly compare data with simulation in a large number of exclusive final states (bins).

Recent proposals extend this methodology to be unbinned by using nearest neighbors [54], cluster similarity [55] and neural networks [56, 57]. Another innovation in [56] is the introduction of a new

loss function that leads the neural network to learn the log likelihood ratio directly:

$$\mathcal{L}(f) = \sum_i (1 - y_i)(e^{f(x_i)} - 1) - y_i f(x_i). \quad (2)$$

In the asymptotic limit (sufficient training data and network/training flexibility), one could then interpret the neural network output as log likelihood ratio which has analytic formulae for computing p -values [58–60]. Interpreting the output directly as a test statistic/likelihood ratio estimator has also been used for a variety of other studies related to simulation-based inference and domain adaptation [17, 61–70].

The approaches mentioned above are nearly signal model independent. The only signal model dependence is in the selection of the targeted phase space and analysis features. This is particularly beneficial for non-resonant new physics,^c where there are fewer methods available. However, there is a strong dependence on the background model. Any significant mis-modeling of the background spectrum will be flagged as an anomaly. Therefore, potential signals need to be larger than these mis-modelings and the false positive rate must be calibrated to account for the expected deviations between data and simulation.

4. Supervised Approaches

The remainder of this chapter focuses on searches that do not use differences between background simulations and data to directly achieve signal sensitivity. Many of the following methods still heavily rely on background simulations for the most natural background calibration method, but that is not discussed further here. In general, techniques can be classified based on their level of *supervision*. Fully supervised approaches use data with labels of signal and background for the y_i in their loss functions. Unsupervised approaches have no per-instance labels and must resort to other methods of identifying structure in

^cThis may not apply to cases of strong signal-background interference.

data. A variety of methods called *weakly supervised* are based on incomplete label information and will be introduced in Sec. 6.

One strategy for supervised anomaly detection is to train with a signal simulation that includes a variety of individual signal processes. This idea was explored in one of the first modern machine learning based anomaly detection procedures called *anti-QCD tagging* [38] (see also [71]). Instead of using a discrete set of signal models [72], a signal dataset was generated using flat matrix elements. This means that there is no special energy scale in the problem and there is broad coverage for all of the kinematic configurations that can occur with a certain number of final state objects. In this way, no particular signal masses were preferred, but the classifier was able to learn generic features of a large class of signal models such as the number of prongs within a jet. A generic feature of anomaly detection that was well-characterized in [38] is that these approaches are less sensitive than dedicated searches for targeted signal models. However, anomaly detection methods can be more sensitive than dedicated searches for non-targeted signal models.

5. Unsupervised Approaches

Unsupervised methods do not use per-instance labels. The strategy of these approaches is to identify events that are unlike typical background events. One way to do this would be to learn the density $p_{\text{background}}(x)$ and then consider events with $p_{\text{background}}(x) \ll 1$. Learning high-dimensional densities is difficult and so this has not yet been studied.^d Instead, the approaches that have been studied so far use compression methods (see also *representation learning* [73]). Two networks f and g encode and decode data, respectively. These networks are trained to be near inverses of each other: $f(g(x)) \approx x$. The target of f is regularized so that the compression is lossy. The idea is that events with $f(g(x)) \approx x$ should be common and thus

^dDensity estimation has been studied in the context of likelihood ratio estimation [16].

ignored while events with $f(g(x))$ far from x are anomaly candidates. This strategy has been studied in the context of autoencoders (AE)/variational autoencoders (VAE [74, 75]) [76–82] and generative adversarial networks (GANs [83]) [84]. Figure 2 illustrates the method for vanilla autoencoders. The lossy compression is achieved by limiting the expressivity of f and the dimensionality of $f(x)$. Both f and g are trained at the same time when minimizing the loss $|f(g(x)) - x|^2$. A classifier is created using the loss (“reconstruction error”). Ideally, typical events should have a lower reconstruction loss than events that were not used or at least minimally present in the training of the autoencoder. This is explicitly demonstrated in the bottom plot of Fig. 2, where the more typical generic QCD jet processes have a much lower reconstruction loss than top quark pair production or gluino production. Other strategies for regulating the latent space are possible, such as requiring it to be of a particular form. Variational autoencoders (VAEs) use this strategy, where the latent space is typically required to be a multivariate Gaussian distribution. The anomaly detection based on a GAN presented in [84] is conceptually similar to the way AEs are used for anomaly detection, but the model is trained using a bidirectional GAN [85] instead of a variational autoencoder setup. Another unsupervised approach using a clustering method was proposed in [86].

A challenge with unsupervised approaches is that the signal may not occupy regions of low $p_{\text{background}}$. If instead, the signal is an over-density in a region of relatively high background probability density, then it may be well-reconstructed by compression methods. Additionally, a feature of some compression methods is that high reconstruction loss may not necessarily correspond to low $p_{\text{background}}$. For example, consider a simple autoencoder trained to compress two dimensions into one dimension. Further suppose that the background is a bivariate Gaussian. A limited capacity network may simply learn $f(x_0, x_1) = \rho \frac{\sigma_1}{\sigma_0} (x_0 - \mu_0) + \mu_1$, where μ_i, σ_i are the mean and standard deviation of the i th direction and ρ is the correlation coefficient. This function is the minimum variance unbiased estimator of x_1 given x_0 . If the signal is also a Gaussian with mean far from the line defined by $f(x)$, then it will have a poor reconstruction error. However, the

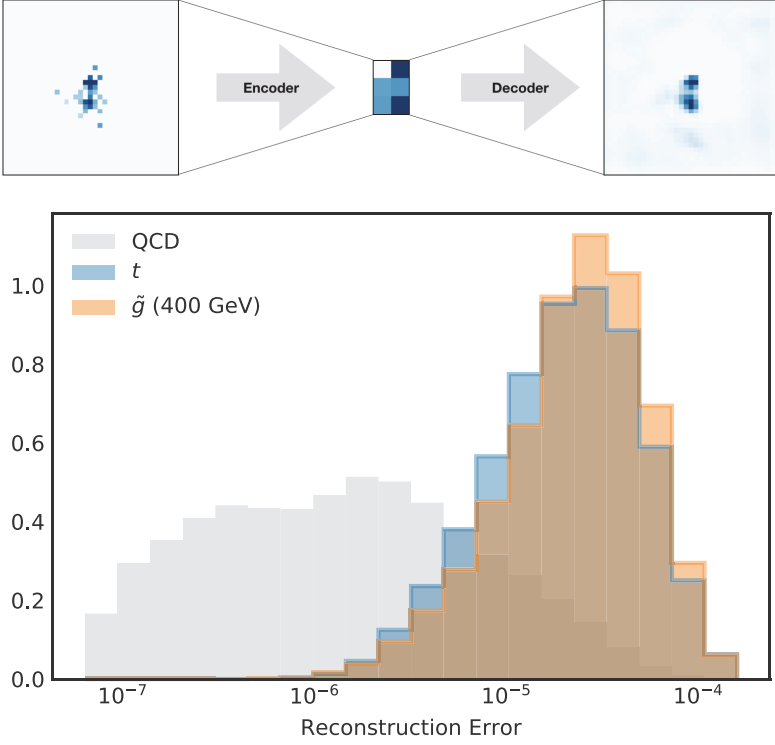


Fig. 2. An illustration of a vanilla autoencoder-based anomaly detection strategy. The top diagram illustrates how the image of a jet is compressed and then restored with two neural networks. The bottom plot is the reconstruction error $|f(g(x)) - x|^2$, which is higher for processes that were not used or at least minimally present in the training of the autoencoder. Figure reproduced from [76].

signal could also be on the line, but from the (μ_0, μ_1) . This would correspond to a very low $p_{\text{background}}$, but also a low reconstruction error. Despite these challenges, AEs are popular in HEP and beyond (see [87] and papers that cite it) and are sufficiently generic that they may be complementary to the approaches described in the next sections.

6. Weak Supervision and Topic Modeling

One challenge with the unsupervised methods in the previous section is that they do not explicitly use the (potential) presence of signal in

the data. This section will introduce methods that make use of the presence of the signal by using supervised learning without explicit per instance labels. One setting where this is possible is the case of mixed samples. Suppose that there are two sets of data \mathcal{M}_0 and \mathcal{M}_1 , each of which is a mixture of S and B . If the per-instance labels are known, then one can train a fully supervised classifier. However, if these sets are from real events, then per-instance labels are unknown. A variety of weakly supervised methods have been developed for this setting [18, 88–90]. The first of these posited that the fractions t of signal in each dataset are known. Then, one can train a weakly supervised classifier:

$$f_{\text{weak}} = \operatorname{argmin}_{f': \mathbb{R}^n \rightarrow [0,1]} \mathcal{L} \left(\sum_{i=1}^N \frac{f'(x_i)}{N} - t \right), \quad (3)$$

which should be contrasted with the fully supervised case:

$$f_{\text{supervised}} = \operatorname{argmin}_{f': \mathbb{R}^n \rightarrow [0,1]} \sum_{i=1}^N \mathcal{L} (f'(x_i) - y_i), \quad (4)$$

where in both cases, \mathcal{L} is a loss function such as mean squared error. This modification to the loss is effective, but (a) requires significant modification to the learning setup (learn on average instead of per-instance) and (b) requires the fractions to be known.^e An alternative approach is the classification without labels (CWoLa) [18] framework. In this setup, one *assigns* labels to each event and then performs supervised learning with the assigned labels. In particular, all events from \mathcal{M}_0 are labeled 0 and all the events in \mathcal{M}_1 are labeled 1. This is illustrated in the left part of Fig. 3. One can show that if the CWoLa classifier is optimal for distinguishing \mathcal{M}_0 from \mathcal{M}_1 , then it will be also optimal at distinguishing^f S from B . The right plot of Fig. 3 shows that the CWoLa classifier achieves the same performance as

^eThe label fractions need not be known exactly for optimal performance [90].

^fIf $0 \leftrightarrow 1$, then one may learn to anti-tag the signal. This simply then requires that one know which of \mathcal{M}_0 or \mathcal{M}_1 is expected to have a higher signal proportion.

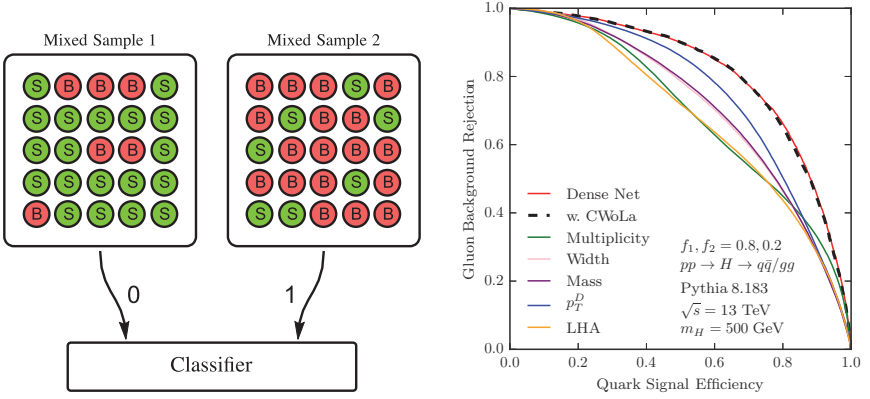


Fig. 3. The left diagram illustrates the setup of the CWoLa method and the right plot demonstrates that the weakly supervised CWoLa can achieve the same performance as a fully supervised classifier trained with the same features. In the right plot, a better classifier would be up and to the right (gluon jet rejection is one minus the gluon jet efficiency). Figure reproduced from [18].

a fully supervised classifier on a particular quark-jet vs. gluon-jet classification task. This is particularly powerful because the fully supervised classifier has per instance labels y_i while the CWoLa classifier has far less information. Before proceeding, it is important to consider the assumptions of the CWoLa method. Most importantly, this approach only works if the differences between $p(S|\mathcal{M}_0)$ and $p(S|\mathcal{M}_1)$ (similarly for the background) are small compared to the difference between $p(S|\mathcal{M}_0)$ and $p(B|\mathcal{M}_0)$. In other words, the signal events from \mathcal{M}_0 and \mathcal{M}_1 should be statistically identical (and the same for the background). The only other requirement is that the signal fractions should not be the same. The effective statistics available to learn scale like $n_{\text{signal}}(y_0 - y_1)$ so the closer the fractions y_0 and y_1 are to each other, the worse the classification performance will be. In the limit $y_0 \rightarrow 0$ and $y_1 \rightarrow 1$, CWoLa simply approaches fully supervised classification. Note that in order to make the performance curve in Fig. 3, one needs at least a small set of labeled examples or the class fractions. If one does not need to know the actual performance, this is not necessary.

A variety of related concepts have been introduced for similar purposes. The sPlot [91, 92] method learns to decompose a dataset into its constituent processes without per-instance labels, but it requires knowing $p(x|S)$ and $p(x|B)$ for at least a subset of the features. The topic modeling introduced in [93] and further studied in [93–96] relaxes this requirement by extracting information directly from data using extreme regions of phase space that are nearly pure S or B . Another variation that seeks to solve the challenge of different classes sharing common features is the Latent Dirichlet Allocation (LDA) [14] approach to mixed-membership models in [15, 97].

There are many ways to combine the weakly supervised methods described above with anomaly detection. One approach combines CWoLa with a bump hunt in a feature m_{res} as illustrated in Fig. 4. In particular, two mixed samples are constructed by using a signal region around a hypothetical resonance and a sideband region. The sideband region should be as close as possible to the signal region in order to ensure that the background is nearly the same in the two mixed samples. The other features used for training the CWoLa classifier need to be nearly independent of m_{res} . The bottom plots in Fig. 4 illustrates the performance of this “CWoLa hunting” method to a dijet search at the LHC using the dijet mass as m_{res} and other jet substructure features to train the CWoLa classifier. In the absence of an injected signal, the p -values are consistent with uniform while when there is signal injected, the initially 1.5σ excess is automatically enhanced to a 7σ potential discovery. In order to make the most use of the data and to avoid a large trials factor, this search involves a complex k -fold cross-training procedure. The look elsewhere effect would be significant if the same data were used for training and applying the CWoLa classifier as local fluctuations would be amplified. One way around this is to divide the data in half and train one one part and test on the other. The local fluctuations in both halves are uncorrelated and thus there is no additional trials factor aside from the one in the scan of m_{res} . This procedure is extended to k -fold in [19, 20] to avoid using only half of the data.

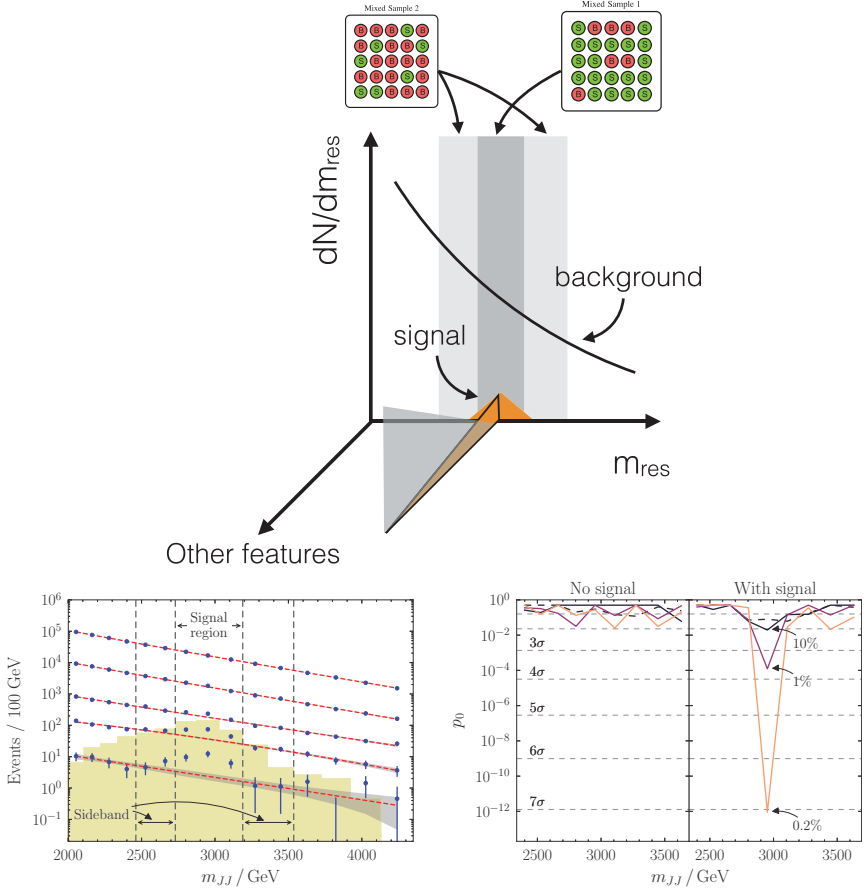


Fig. 4. Top: A schematic diagram illustrating the CWoLa setup for the anomaly detection task. Bottom: Figures reproduced from [19, 20] that show the application of the CWoLa hunting method to the dijet search at the LHC. The bottom left plot is the dijet invariant mass with the different lines corresponding to increasingly tighter thresholds on the CWoLa classifier. The injected signal is many order of magnitude below the background. The bottom right plot shows the extracted p -value for a scan in the signal region location. Without any injected signal, the p -values are consistent with uniform while when there is signal injected, the initially 1.5σ excess is automatically enhanced to a 7σ potential discovery.

7. Hybrid Approaches

One of the main limitations of the CWoLa hunting method is that the features used for training the CWoLa classifier must be nearly independent of m_{res} . One hybrid solution proposed to mitigate this problem is ANODE [16]. In this method, one first learns $p_{\text{sideband}}(x|m_{\text{res}})$ in the sideband regions and then $p_{\text{signal region}}(x|m)$ from the signal region. The ratio of the interpolated $p_{\text{sideband}}(x|m_{\text{res}})$ to $p_{\text{signal region}}(x|m)$ serves as an asymptotically optimal likelihood ratio estimator. There have been significant advances in direct density estimation and the demonstration of ANODE in [16] makes use of masked autoregressive flows [98], a type of normalizing flow [99]. Density learning is unsupervised because there are no labels and one typically learns p via a maximum likelihood loss function. The main challenge in direct density estimation is that one needs a neural network that integrates to unity. In the normalizing flows setting, this is accomplished by starting with a known density (often a Gaussian) and then applying a series of variable changes with tractable Jacobians. In ANODE, one never compares signal region and sideband region directly and so it is more robust to correlations between x and m_{res} ; in fact, m_{res} can be one of the dimensions of x . This is illustrated in Fig. 5, which shows that when correlations are artificially introduced to spoil the CWoLa hunting approach, ANODE is still able to provide signal sensitivity. With advances in neural density estimation (see e.g. Neural Autoregressive Flows [100] and Neural Spline Flows [101]), it is likely that methods like ANODE will continue to improve. A key difference for HEP compared to industrial applications is that one needs quantitatively precise density estimation — it is not good enough to produce examples that qualitatively look realistic (as in the case of cat and celebrity pictures — see e.g. <https://thispersondoesnotexist.com>).

Another hybrid solution is Simulation Assisted Likelihood-free Anomaly Detection (SALAD) [17] (see also SA-CWoLa [102]). The motivation of this method is that while simulations are only an approximation to nature, they do encode significant physics

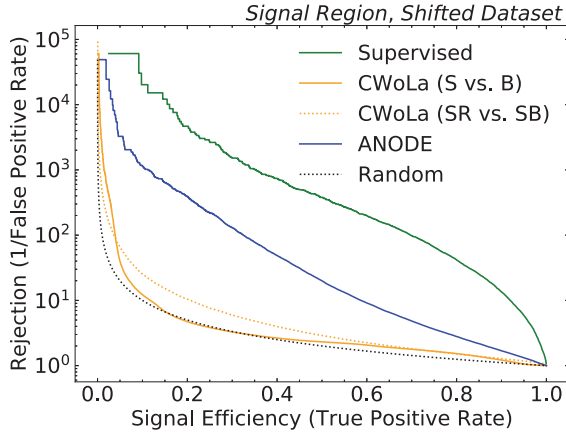


Fig. 5. The performance of CWoLa and ANODE classifiers on a dataset with artificial correlations. The CWoLa classifier learns a non-trivial function for signal region vs. sideband region and as a result has only random performance on the signal. In contrast, by learning the numerator and denominator of the likelihood ratio separately, ANODE is more robust to such correlations (but still worse than the dedicated fully supervised tagger). Figure reproduced from [16].

information that it would be useful to incorporate into a classifier. To this end, a parameterized reweighting function $w(x|m) \approx p_{\text{data}}(x|m)/p_{\text{simulation}}(x|m)$ is learned in sidebands and then interpolated to the signal region [61]. Any classifier can then be used to distinguish the reweighted simulation in the signal region from the data in that region. If the reweighting is effective, the result should not depend on the simulation. At the same time, the closer the initial simulation is to the data, the less reliant the method is on an effective learning and interpolation for $w(x|m)$.

A third hybrid method is Tag N' Train (TNT) [103], which combined autoencoders with CWoLa. The motivation is that in the original CWoLa hunting method, the two mixed samples are nearly 100% of a single class (background) so as a first step, a weak classifier (using an autoencoder — see Sec. 5) slightly purifies the samples before using CWoLa to train a more powerful classifier. As discussed in Sec. 6, the purer the samples, the larger the effective statistics for the CWoLa training and thus the more powerful it will be as a classifier.

It is likely that no one method will be able to cover every scenario and so a diversity of approaches will be needed to ensure broad coverage. Mixing different approaches may provide further advantages over single algorithms using either entirely supervised or entirely un/weakly supervised methods.

8. Results with Collider Data

This chapter will close with a highlight of two recent results from the CMS and ATLAS Collaborations. Figure 6 presents the first application of CWoLa^g in a physics analysis. The CWoLa classifier is used to distinguish generic multijet events from $t\bar{t}$ production and uses

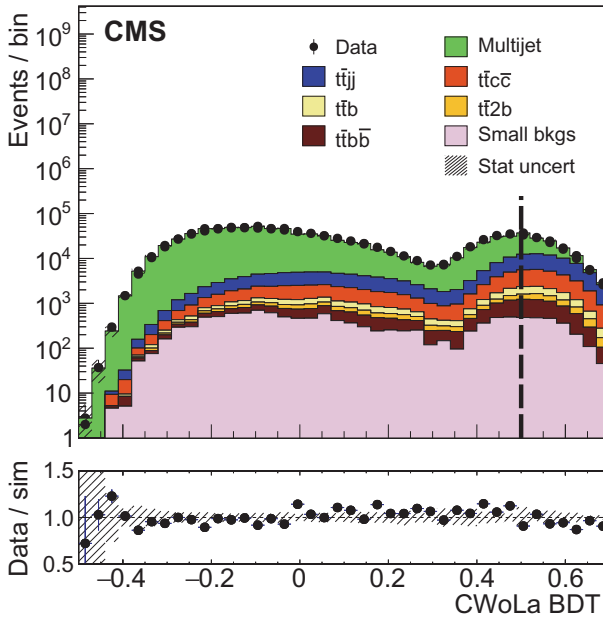


Fig. 6. A CWoLa classifier (Boosted Decision Tree) used to purify the $t\bar{t} + b\bar{b}$ final state by the CMS Collaboration. Figures reproduced from [104].

^gSee [95] for the first measurement to study the related idea of topic modeling.

two regions with $\mathcal{O}(10\%)$ signal purity. The ultimate goal of this analysis is to study the $t\bar{t} + b\bar{b}$ process that is an inherently interesting probe of the strong force and is also an important process for Higgs physics in the $t\bar{t} + h$ final state. Multijet backgrounds are exceptionally difficult to simulate accurately and the CWoLa classifier provides a solution to train directly from data. While not an anomaly detection search, this analysis demonstrates the feasibility of learning directly from unlabeled data.

The first machine learning anomaly detection result with data is presented in Fig. 7 from the ATLAS Collaboration. This search made use of the CWoLa hunting approach described in Sec. 6 and targets a dijet final state. This final state is complex and challenging to model, so data-driven methods are critical. When the jets are due to hadronically decaying particles that are much lighter than the target resonance mass, their decay products will be collimated and fully contained inside single jets. The substructure of these jets can be exploited to enhance the signal sensitivity.

As the first search of its kind, a limited feature space (two-dimensional) was used to help with the technical and sociological integration of this new methodology into the experimental program. With a two-dimensional feature space, the classifier output can be directly visualized as an image. Examples of these images are shown in the top plots of Fig. 7, where the automatic identification of an injected signal is demonstrated. The bottom plot of Fig. 7 shows that for particular models, the CWoLa hunting analysis is able to set the strongest limits for particular models. One of the most interesting challenges that is like no other analysis is that the event selection depends on the data. This means that for every injected signal model, for every signal strength, the entire CWoLa hunting procedure has to be rerun. Aside from making it challenging to recast this analysis, a large number of neural networks ($\mathcal{O}(10^4)$) must be trained to produce Fig. 7.

The methods discussed in this section have not yet resulted in the detection of an anomaly in collider data at the time of writing, but this is only the beginning of an expanding program that will produce exciting physics results for many years to come.

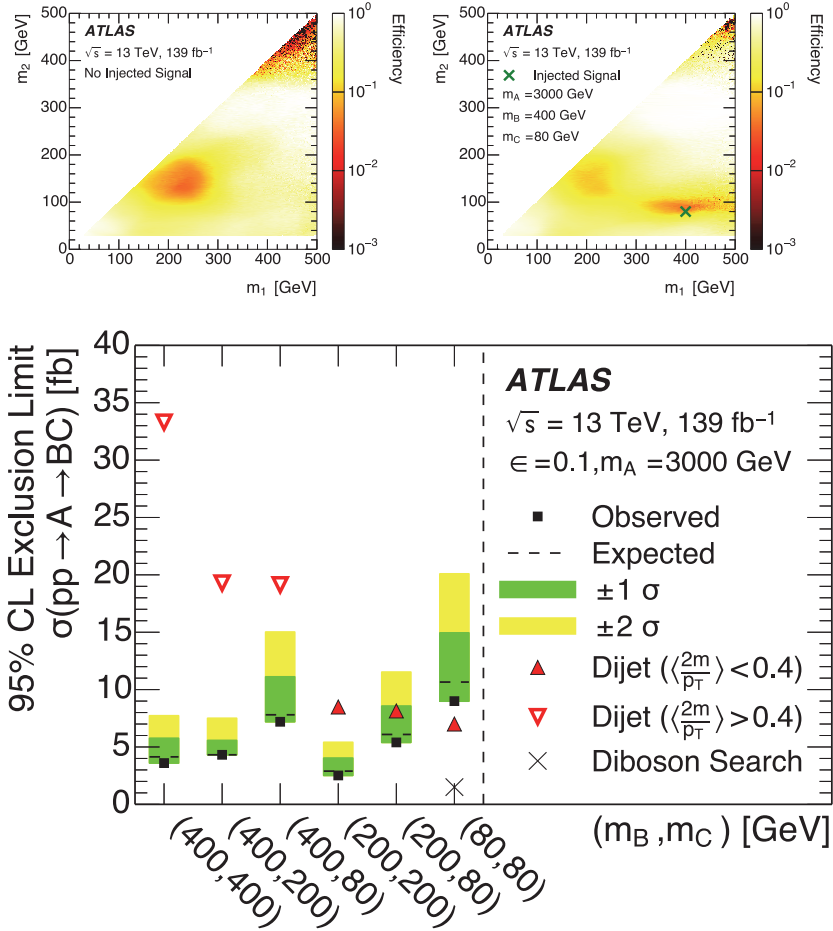


Fig. 7. Plots from the recent CWoLa hunting search performed by the ATLAS Collaboration [105]. The top two plots show the CWoLa classifier output in one particular signal region on the left with no injected signal and with an injected signal with masses indicated by the cross in the right plot. The CWoLa classifier is able to automatically identify the signal-like region. While there was no significant evidence for new particles in this first search, the ATLAS Collaboration has set limits on the production cross section for particular models.

9. Conclusions and Outlook

Given the current lack of convincing evidence for new fundamental structure from HEP experiments, it is critical that the program

of dedicated searches be complemented with more model agnostic methods. The methods presented in this chapter represent a snapshot^h of the rapidly developing area of machine learning for anomaly detection in HEP.

To help catalyze new ideas in anomaly detection for HEP,ⁱ the LHC Olympics 2020 was developed [108]. This data challenge sets up a realistic environment where there is “simulation” and “data”, where both are generated on a computer, but possibly from different physics programs. Furthermore, the “data” comes without labels and may contain some amount of signal. A variety of methods have been deployed to these datasets, exposing interesting features of the various procedures and helping to prepare for analysis with real data. With the first results from collider data presented in Sec. 8, the field enters a new era where methods must be adapted and modified to meet the needs of real data and new methods must be developed to ensure broad coverage.

Addressing the conceptual, computational, and other challenges associated with the growing field of machine learning-assisted anomaly detection in HEP will make for an exciting research programs in the years ahead.

Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science under contract DE-AC02-05CH11231. I thank Kees Benkendorfer, Jack Collins, Aviv Cukierman, Gregor Kasieczka, Luc Le Pottier, and David Shih for countless discussions about anomaly detection that have contributed to the framing presented in this

^hSee [106] for a more updated list of papers in this area.

ⁱA parallel effort under development is described in [107]. This dataset is planned to be a concoction of SM processes and uses high-level objects instead of low-level hadrons as in the LHC Olympics. While the current plan does not include multiple generators to emulate “data” and “simulation”, the large number of events and heterogeneous composition of physics processes offers an interesting complement to the LHC Olympics.

chapter. Furthermore, I thank Gregor Kasieczka for detailed comments on the manuscript. I also thank Anders Andreassen, Patrick Komiske, Eric Metodiev, Matt Schwartz, and Jesse Thaler for many helpful discussions about learning from mixed samples.

References

- [1] **CMS** Collaboration, S. Chatrchyan *et al.*, Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC, *Phys. Lett. B* **716** (2012) 30–61; arXiv:1207.7235.
- [2] **ATLAS** Collaboration, G. Aad *et al.*, Observation of a new particle in the search for the standard model Higgs boson with the ATLAS detector at the LHC, *Phys. Lett. B* **716** (2012) 1–29; arXiv:1207.7214.
- [3] A. L. Read, Presentation of search results: The CL_s technique, *J. Phys. G* **28** (2002) 2693–2704.
- [4] J. Neyman and E. S. Pearson, On the problem of the most efficient tests of statistical hypotheses, *Phil. Trans. R. Soc. Lond. A* **231** (1933) 289.
- [5] B. Nachman, A guide for deploying deep learning in LHC searches: How to achieve optimality and account for uncertainty, *SciPost Phys.* **8** (2020) 090; arXiv:1909.03081.
- [6] P. De Castro and T. Dorigo, INFERNO: Inference-aware neural optimisation, *Comput. Phys. Commun.* **244** (2019) 170–179; arXiv:1806.04743.
- [7] S. Wunsch, S. Jöger, R. Wolf and G. Quast, Optimal statistical inference in the presence of systematic uncertainties using neural network optimization based on binned Poisson likelihoods with nuisance parameters, preprint (2020) arXiv:2003.07186.
- [8] **CMS** Collaboration, MUSiC: A model unspecific search for new physics, in pp collisions at $\sqrt{s} = 8$ TeV, CMS-PAS-EXO-14-016 (2017).
- [9] **CMS** Collaboration, Model unspecific search for new physics in pp collisions at $\sqrt{s} = 7$ TeV, CMS-PAS-EXO-10-021 (2011).
- [10] **CMS** Collaboration, MUSiC: A model unspecific search for new physics in proton-proton collisions at $\sqrt{s} = 13$ TeV, preprint (2020); arXiv:2010.02984.
- [11] **ATLAS** Collaboration, M. Aaboud *et al.*, A strategy for a general search for new phenomena using data-derived signal regions and its application within the ATLAS experiment, *Eur. Phys. J.* **C79** (2019) 120; arXiv:1807.07447.
- [12] **ATLAS** Collaboration, A general search for new phenomena with the ATLAS detector in pp collisions at $\sqrt{s} = 8$ TeV, ATLAS-CONF-2014-006 (2014).
- [13] **ATLAS** Collaboration, A general search for new phenomena with the ATLAS detector in pp collisions at $\sqrt{s} = 7$ TeV, ATLAS-CONF-2012-107 (2012).
- [14] D. M. Blei, A. Y. Ng and M. I. Jordan, Latent Dirichlet allocation, *J. Mach. Learn. Res.* **3** (2003) 993–1022.

- [15] B. M. Dillon, D. A. Faroughy and J. F. Kamenik, Uncovering latent jet substructure, *Phys. Rev. D* **100**(5) (2019) 056002; arXiv:1904.04200.
- [16] B. Nachman and D. Shih, Anomaly detection with density estimation, *Phys. Rev. D* **101** (2020) 075042; arXiv:2001.04990.
- [17] A. Andreassen, B. Nachman and D. Shih, Simulation assisted likelihood-free anomaly detection, *Phys. Rev. D* **101**(9) (2020) 095004; arXiv:2001.05001.
- [18] E. M. Metodiev, B. Nachman and J. Thaler, Classification without labels: Learning from mixed samples in high energy physics, *J. High Energy Phys.* **10** (2017) 174; arXiv:1708.02949.
- [19] J. H. Collins, K. Howe and B. Nachman, Anomaly detection for resonant new physics with machine learning, *Phys. Rev. Lett.* **121**(24) (2018), 241803; arXiv:1805.02664.
- [20] J. H. Collins, K. Howe and B. Nachman, Extending the search for new resonances with machine learning, *Phys. Rev. D* **99**(1) (2019), 014038; arXiv:1902.02634.
- [21] **ATLAS** Collaboration, M. Aaboud *et al.*, Search for heavy ZZ resonances in the $\ell^+\ell^-\ell^+\ell^-$ and $\ell^+\ell^-\nu\bar{\nu}$ final states using proton–proton collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector, *Eur. Phys. J. C* **78**(4) (2018), 293, arXiv:1712.06386.
- [22] G. Kasieczka, B. Nachman, M. D. Schwartz and D. Shih, ABCDisCo: Automating the ABCD method with machine learning, preprint (2020); arXiv:2007.14400.
- [23] S. Choi, J. Lim and H. Oh, Data-driven estimation of background distribution through neural autoregressive flows, preprint (2020); arXiv:2008.03636.
- [24] **ATLAS** Collaboration, Search for Higgs boson decays into a Z boson and a light hadronically decaying resonance using 13 TeV pp collision data from the ATLAS detector, preprint (2020); arXiv:2004.01678.
- [25] M. Frate, K. Cranmer, S. Kalia, A. Vandenberg-Rodes and D. Whiteson, Modeling smooth backgrounds and generic localized signals with Gaussian processes, preprint (2017); arXiv:1709.05681.
- [26] R. Di Sipio, M. Fauci Giannelli, S. Ketabchi Haghighat and S. Palazzo, DijetGAN: A generative-adversarial network approach for the simulation of QCD dijet events at the LHC, *J. High Energy Phys.* **19** (2020) 110; arXiv:1903.02433.
- [27] G. Louppe, M. Kagan, and K. Cranmer, Learning to pivot with adversarial networks, *Advances in Neural Information Processing Systems* **30** (2017); arXiv:1611.01046.
- [28] J. Dolen, P. Harris, S. Marzani, S. Rappoccio and N. Tran, Thinking outside the ROCs: Designing decorrelated taggers (DDT) for jet substructure, *J. High Energy Phys.* **05** (2016) 156; arXiv:1603.00027.
- [29] I. Moulst, B. Nachman and D. Neill, Convolved substructure: Analytically decorrelating jet substructure observables, *J. High Energy Phys.* **05** (2018) 002; arXiv:1710.06859.
- [30] J. Stevens and M. Williams, uBoost: A boosting method for producing uniform selection efficiencies from multivariate classifiers, *JINST* **8** (2013) P12013; arXiv:1305.7248.

- [31] C. Shimmin, P. Sadowski, P. Baldi, E. Weik, D. Whiteson, E. Goul and A. Sogaard, Decorrelated jet substructure tagging using adversarial neural networks, *Phys. Rev. D* **96**(7) (2017) 074034; arXiv:1703.03507.
- [32] L. Bradshaw, R. K. Mishra, A. Mitridate and B. Ostdiek, Mass agnostic jet taggers, *SciPost Phys.* **8**(1) (2020) 011; arXiv:1908.08959.
- [33] **ATLAS** Collaboration, Performance of mass-decorrelated jet substructure observables for hadronic two-body decay tagging in ATLAS, *ATL-PHYS-PUB-2018-014* (2018).
- [34] G. Kasieczka and D. Shih, Robust jet classifiers through distance correlation, *Phys. Rev. Lett.* **125**(12) (2020) 122001; arXiv:2001.05310.
- [35] L.-G. Xia, QBDT: A new boosting decision tree method with systematical uncertainties into training for high energy physics, *Nucl. Instrum. Meth. A* **930** (2019) 15–26; arXiv:1810.08387.
- [36] C. Englert, P. Galler, P. Harris and M. Spannowsky, Machine learning uncertainties with adversarial neural networks, *Eur. Phys. J. C* **79**(1) (2019) 4; arXiv:1807.08763.
- [37] S. Wunsch, S. Jörger, R. Wolf and G. Quast, Reducing the dependence of the neural network function to systematic uncertainties in the input space, *Comput. Softw. Big Sci.* **4**(1) (2020) 5; arXiv:1907.11674.
- [38] J. A. Aguilar-Saavedra, J. H. Collins and R. K. Mishra, A generic anti-QCD jet tagger, *J. High Energy Phys.* **11** (2017) 163; arXiv:1709.01087.
- [39] **CMS** Collaboration, Identification of heavy, energetic, hadronically decaying particles using machine-learning techniques, *JINST* **15**(06) (2020) P06005; arXiv:2004.08262.
- [40] O. Kitouni, B. Nachman, C. Weisser and M. Williams, Enhancing searches for resonances with machine learning and moment decomposition, preprint (2020); arXiv:2010.09745.
- [41] **ATLAS** Collaboration, Search for new resonances in mass distributions of jet pairs using 139 fb⁻¹ of *pp* collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector, *J. High Energy Phys.* **03** (2020) 145; arXiv:1910.08447.
- [42] **CMS** Collaboration, Search for high mass dijet resonances with a new background prediction method in proton–proton collisions at $\sqrt{s} = 13$ TeV, *J. High Energy Phys.* **05** (2020) 033; arXiv:1911.03947.
- [43] B. Knuteson, A quasi-model-independent search for new high p_T physics at D0, Ph.D. thesis, University of California at Berkeley (2000).
- [44] **D0** Collaboration, B. Abbott *et al.*, Search for new physics in $e\mu X$ data at D0 using Sherlock: A quasi model independent search strategy for new physics, *Phys. Rev. D* **62** (2000) 092004; arXiv:hep-ex/0006011.
- [45] **D0** Collaboration, V. M. Abazov *et al.*, A Quasi model independent search for new physics at large transverse momentum, *Phys. Rev. D* **64** (2001) 012004; arXiv:hep-ex/0011067.
- [46] **D0** Collaboration, B. Abbott *et al.*, A quasi-model-independent search for new high p_T physics at D0, *Phys. Rev. Lett.* **86** (2001) 3712–3717; arXiv:hep-ex/0011071.
- [47] **H1** Collaboration, F. D. Aaron *et al.*, A general search for new phenomena at HERA, *Phys. Lett. B* **674** (2009) 257–268; arXiv:0901.0507.

- [48] **H1** Collaboration, A. Aktas *et al.*, A general search for new phenomena in ep scattering at HERA, *Phys. Lett. B* **602** (2004) 14–30; arXiv:hep-ex/0408044.
- [49] K. S. Cranmer, Searching for new physics: Contributions to LEP and the LHC. Ph.D. thesis, Wisconsin University, Madison (2005).
- [50] **CDF** Collaboration, T. Aaltonen *et al.*, Model-independent and quasi-model-independent search for new physics at CDF, *Phys. Rev. D* **78** (2008) 012002; arXiv:0712.1311.
- [51] **CDF** Collaboration, T. Aaltonen *et al.*, Model-independent global search for new high-p(T) physics at CDF, preprint (2007); arXiv:0712.2534.
- [52] **CDF** Collaboration, T. Aaltonen *et al.*, Global search for new physics with 2.0 fb⁻¹ at CDF, *Phys. Rev. D* **79** (2009) 011101; arXiv:0809.3781.
- [53] **CMS** Collaboration, MUSIC: A model unspecific search for new physics in proton–proton collisions at $\sqrt{s} = 13$ TeV, preprint (2020); arXiv:2010.02984.
- [54] A. De Simone and T. Jacques, Guiding new physics searches with unsupervised learning, *Eur. Phys. J. C* **79**(4) (2019) 289; arXiv:1807.06038.
- [55] G. M. Alessandro Casa, Nonparametric semisupervised classification for signal detection in high energy physics, preprint (2018); arXiv:1809.02977.
- [56] R. T. D’Agnolo and A. Wulzer, Learning new physics from a machine, *Phys. Rev. D* **99**(1) (2019) 015014; arXiv:1806.02350.
- [57] R. T. D’Agnolo, G. Grosso, M. Pierini, A. Wulzer and M. Zanetti, Learning multivariate new physics, preprint (2019); arXiv:1912.12155.
- [58] S. S. Wilks, The large-sample distribution of the likelihood ratio for testing composite hypotheses, *Ann. Math. Statist.* **9** (1938) 60–62.
- [59] G. Cowan, K. Cranmer, E. Gross and O. Vitells, Asymptotic formulae for likelihood-based tests of new physics, *Eur. Phys. J. C* **71** (2011) 1554; arXiv:1007.1727. [Erratum: *Eur. Phys. J. C* **73** (2013) 2501].
- [60] A. Wald, Tests of statistical hypotheses concerning several parameters when the number of observations is large, *Trans. Amer. Math. Soc.* **54**(3) (1943), 426–482.
- [61] A. Andreassen and B. Nachman, Neural networks for full phase-space reweighting and parameter tuning, *Phys. Rev. D* **101**(9) (2020), 091901; arXiv:1907.08209.
- [62] M. Stoye, J. Brehmer, G. Louppe, J. Pavez and K. Cranmer, Likelihood-free inference with an improved cross-entropy estimator, preprint (2018); arXiv:1808.00973.
- [63] J. Hollingsworth and D. Whiteson, Resonance searches with machine learned likelihood ratios, preprint (2020); arXiv:2002.04699.
- [64] J. Brehmer, K. Cranmer, G. Louppe and J. Pavez, Constraining effective field theories with machine learning, *Phys. Rev. Lett.* **121**(11) (2018), 111801; arXiv:1805.00013.
- [65] J. Brehmer, K. Cranmer, G. Louppe and J. Pavez, A guide to constraining effective field theories with machine learning, *Phys. Rev. D* **98**(5) (2018), 052004; arXiv:1805.00020.

- [66] J. Brehmer, F. Kling, I. Espejo and K. Cranmer, MadMiner: Machine learning-based inference for particle physics, *Comput. Softw. Big Sci.* **4**(1) (2020) 3; arXiv:1907.10621.
- [67] J. Brehmer, G. Louppe, J. Pavez and K. Cranmer, Mining gold from implicit models to improve likelihood-free inference, *Proc. Nat. Acad. Sci. USA* **117**(10) (2020) 5242–5249; arXiv:1805.12244.
- [68] K. Cranmer, J. Pavez and G. Louppe, Approximating likelihood ratios with calibrated discriminative classifiers, preprint (2015); arXiv:1506.02169.
- [69] C. Badiali, F. Di Bello, G. Frattari, E. Gross, V. Ippolito, M. Kado and J. Shlomi, Efficiency parameterization with neural networks, preprint (2020); arXiv:2004.02665.
- [70] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman and J. Thaler, OmniFold: A method to simultaneously unfold all observables, *Phys. Rev. Lett.* **124**(18) (2020) 182001; arXiv:1911.09107.
- [71] J. Aguilar-Saavedra, F. Joaquim and J. Seabra, Mass unspecific supervised tagging (MUST) for boosted jets, preprint (2020); arXiv:2008.12792.
- [72] C. K. Khosa and V. Sanz, Anomaly awareness, preprint (2020); arXiv:2007.14462.
- [73] Y. Bengio, A. Courville and P. Vincent, Representation learning: A review and new perspectives, *IEEE Trans. Pattern Anal. Mach. Intell.* **35** (2013) 1798–1828.
- [74] D. P. Kingma and M. Welling, Auto-encoding variational bayes., in *ICLR*, eds. Y. Bengio and Y. LeCun, (2014).
- [75] D. P. Kingma and M. Welling, An introduction to variational autoencoders, *Found. Trends Mach. Learn.* **12**(4) (2019), 307–392.
- [76] M. Farina, Y. Nakai and D. Shih, Searching for new physics with deep autoencoders, *Phys. Rev. D* **101**(7) (2020) 075021; arXiv:1808.08992.
- [77] T. Heimel, G. Kasieczka, T. Plehn and J. M. Thompson, QCD or what? *SciPost Phys.* **6**(3) (2019), 030; arXiv:1808.08979.
- [78] T. S. Roy and A. H. Vijay, A robust anomaly finder based on autoencoder, preprint (2019); arXiv:1903.02032.
- [79] A. Blance, M. Spannowsky and P. Waite, Adversarially-trained autoencoders for robust unsupervised new physics searches, *J. High Energy Phys.* **10** (2019) 047; arXiv:1905.10384.
- [80] J. Hajer, Y.-Y. Li, T. Liu and H. Wang, Novelty detection meets collider physics, *Phys. Rev. D* **101**(7) (2020) 076015; arXiv:1807.10261.
- [81] O. Cerri, T. Q. Nguyen, M. Pierini, M. Spiropulu and J.-R. Vlimant, Variational autoencoders for new physics mining at the large hadron collider, *J. High Energy Phys.* **05**(036); (2019) arXiv:1811.10276.
- [82] T. Cheng, J.-F. Arguin, J. Leissner-Martin, J. Pilette and T. Golling, Variational autoencoders for anomalous jet tagging, preprint (2020); arXiv:2007.01850.
- [83] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, Generative adversarial nets, in *Proc. 27th Int. Conf. Neural Information Processing Systems — Vol. 2, NIPS’14*, (MIT Press, Cambridge, MA, 2014), pp. 2672–2680.

- [84] O. Knapp, G. Dissertori, O. Cerri, T. Q. Nguyen, J.-R. Vlimant and M. Pierini, Adversarially learned anomaly detection on CMS open data: Re-discovering the top quark, preprint (2020); arXiv:2005.01598.
- [85] J. Donahue, P. Krähenbühl and T. Darrell, Adversarial feature learning (2016).
- [86] V. Mikuni and F. Canelli, Unsupervised clustering for collider physics, preprint (2020); arXiv:2010.07106.
- [87] M. A. Pimentel, D. A. Clifton, L. Clifton and L. Tarassenko, A review of novelty detection, *Signal Proces.* **99** (2014) 215–249.
- [88] L. M. Dery, B. Nachman, F. Rubbo, and A. Schwartzman, Weakly supervised classification in high energy physics, *J. High Energy Phys.* **05** (2017) 145; arXiv:1702.00414.
- [89] P. T. Komiske, E. M. Metodiev, B. Nachman and M. D. Schwartz, Learning to classify from impure samples with high-dimensional data, *Phys. Rev. D* **98**(1) (2018) 011502; arXiv:1801.10158.
- [90] T. Cohen, M. Freytsis and B. Ostdiek, (Machine) learning to do more with less, *J. High Energy Phys.* **02** (2018) 034; arXiv:1706.09451.
- [91] M. Pivk and F. R. Le Diberder, SPlot: A statistical tool to unfold data distributions, *Nucl. Instrum. Meth. A* **555** (2005) 356–369; arXiv:physics/0402083.
- [92] M. Borisyak and N. Kazeev, Machine learning on data with sPlot background subtraction, *JINST* **14**(08) (2019), P08020; arXiv:1905.11719.
- [93] E. M. Metodiev and J. Thaler, Jet topics: Disentangling quarks and gluons at colliders, *Phys. Rev. Lett.* **120**(24) (2018) 241602; arXiv:1802.00008.
- [94] P. T. Komiske, E. M. Metodiev and J. Thaler, An operational definition of quark and gluon jets, *J. High Energy Phys.* **11** (2018) 059; arXiv:1809.01140.
- [95] **ATLAS** Collaboration, Properties of jet fragmentation using charged particles measured with the ATLAS detector in pp collisions at $\sqrt{s} = 13$ TeV, *Phys. Rev. D* **100**(5) (2019), 052011; arXiv:1906.09254.
- [96] E. Alvarez, F. Lamagna and M. Szewc, Topic model for four-top at the LHC, *J. High Energy Phys.* **01** (2020) 049; arXiv:1911.09699.
- [97] B. M. Dillon, D. A. Faroughy, J. F. Kamenik and M. Szewc, Learning the latent structure of collider events, preprint (2020); arXiv:2005.12319.
- [98] G. Papamakarios, T. Pavlakou, and I. Murray, Masked autoregressive flow for density estimation, in *Advances in Neural Information Processing Systems 30*, eds. (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett), (Curran Associates, Inc., 2017), pp. 2338–2347.
- [99] D. Rezende and S. Mohamed, Variational inference with normalizing flows, in *Proceedings of Machine Learning Research*, Vol. 37, Lille, France, PMLR, 07–09 July, (2015), pp. 1530–1538.
- [100] C. Huang, D. Krueger, A. Lacoste and A. C. Courville, Neural autoregressive flows, *CoRR abs/1804.00779* (2018); arXiv:1804.00779.
- [101] C. Durkan, A. Bekasov, I. Murray and G. Papamakarios, Neural spline flows, preprint (2019); arXiv:1906.04032.

- [102] L. L. P. K. Benkendorfer and B. Nachman, Simulation-assisted decorrelation for resonant anomaly detection, preprint (2020); arXiv:2009.02205.
- [103] O. Amram and C. M. Suarez, Tag N' Train: A technique to train improved classifiers on unlabeled data, preprint (2020); arXiv:2002.12376.
- [104] **CMS** Collaboration, Measurement of the $t\bar{t}b\bar{b}$ production cross section in the all-jet final state in pp collisions at $\sqrt{s} = 13$ TeV, *Phys. Lett. B* **803** (2020) 135285; arXiv:1909.05306.
- [105] **ATLAS** Collaboration, Dijet resonance search with weak supervision using $\sqrt{s} = 13$ TeV pp collisions in the ATLAS detector, *Phys. Rev. Lett.* **125**(13) (2020) 131801; arXiv:2005.02983.
- [106] M. Feickert and B. Nachman, for the IML Working Group, A living review of machine learning for particle physics, preprint (2020); <https://iml-wg.github.io/HEPML-LivingReview/>.
- [107] G. Brooijmans *et al.*, Les Houches 2019 physics at TeV colliders: New physics working group report, in *11th Les Houches Workshop on Physics at TeV Colliders: PhysTeV Les Houches*, Vol. 2 (2020); arXiv:2002.12220.
- [108] G. Kasieczka, B. Nachman and D. Shih, LHC Olympics 2020, preprint (2020); <https://lhco2020.github.io/homepage/>.

Part II

Data Quality Monitoring

This page intentionally left blank

Chapter 5

Data Quality Monitoring Anomaly Detection

Adrian Alan Pol,^{*,‡} Gianluca Cerminara,^{*,§} Cecile Germain^{†,||}
and Maurizio Pierini^{*,**}

^{*}*CERN*

Espl. des Particules 1, 1211 Meyrin, Switzerland

[†]*LISN, Université Paris-Saclay and CNRS*

91400 Orsay, France

[‡]*adrianalan.pol@cern.ch*

[§]*gianluca.cerminara@cern.ch*

^{||}*cecile.germain@lri.fr*

^{**}*maurizio.pierini@cern.ch*

High energy physics (HEP) experiments involve large and complex detection apparatuses, which need to be operated with high availability to profit at best from the expensive beam time provided by modern particle accelerators. HEP data analysis relies on well understood experimental equipment to chase statistically rare phenomena. These requirements call for constant and reliable monitoring to spot potential malfunctioning. For this reason, HEP collaborations build complex data quality monitoring (DQM) infrastructures and procedures, targeting prompt identification of anomalies arising from either hardware problems or data processing. The HEP monitoring problems are usually challenging for conventional statistical-based methods due to their inherently multidimensional nature. The goal of this chapter is to explore how machine learning (ML) methods of anomaly detection can help improve the existing DQM pipeline, under the specific constraints derived from the critical role of data validation: interpretability of the results and long-term maintainability of the system.

1. Introduction

At the Large Hadron Collider (LHC), the physics data acquisition is a multiple steps process, which involves very complex and large

hardware (both detector and accelerator) and software systems. The detector components collect raw data about the interaction of the particles with the sensitive layers and the trigger system discards the vast majority of bunch collisions. Due to the size and complexity of these systems, transitory or permanent failures of components are unavoidable. Such failures produce erroneous data. Stringent quality criteria must be imposed so that only certifiably good data are passed further to physics analysis. This certification process is called data quality monitoring (DQM) and is a long-established vital procedure in any modern large-scale high energy physics (HEP) experiment.

Failures are not only unavoidable but relatively frequent. For instance, 10%^a of the detector components may manifest problems, while 2%^b of the acquired data is discarded. The relatively high figure of detector components failures is not mostly due to significant, easily detectable, malfunctions of the detector as a whole, but to localized problems. Thanks to the essential redundancy in the detector, in most cases, entirely relevant physics analysis can still be achieved on the data taken by the not-faulty parts of the detector. However, all operational imperfections must be annotated. Thus, a critical goal of the monitoring system, besides the sensitivity, is to be as specific as possible in spotting the defects.

The ever increasing detector complexity and the volume of monitoring data call for a paradigm shift in HEP monitoring, as the techniques in use are swiftly reaching their limits. Machine learning (ML) techniques promise a breakthrough, towards gradually automating the DQM scrutiny and extending the monitoring coverage.

The successes of the neural networks in quality control applications encourage its application to other, more sensitive challenges in HEP, e.g. searches of physics beyond the standard model (Chapter 4).

The goal of this chapter is to explore how ML methods of anomaly detection can help improving the existing DQM pipeline, under the specific constraints derived from the critical role of data validation:

^aCalculations are based on the CMS drift tube sub-detector data [1].

^bCalculations are based on the CMS data certification procedure [2].

interpretability of the results, and long-term maintainability of the system.

To be concrete, we will mostly consider case studies from the CMS DQM. However, the general setting and the challenges are very similar in the other LHC experiments. For more details, see [3–5] for the presentation of ATLAS, LHCb and ALICE DQM procedures.

The chapter presents increasingly difficult use cases. In summary, we show that anomaly detection methods based on deep learning are both efficient and effective. The proposed methods are precisely spotting the problematic data and provide some level of interpretability, making them acceptable to the DQM production system.

The rest of the chapter is organized as follows. Section 2 provides a general outline of the CMS-DQM pipeline, followed by a short survey on ML anomaly detection in Sec. 3. Then, we present the use cases. They can be divided into two groups. In the first one, the main difficulty is to model the problem, based on domain knowledge. Here, standard tools and techniques, i.e. CNNs (Sec. 4.1), deep autoencoders (Secs. 4.2, 4.3 and 5) and LSTMs (Sec. 7) suffice. In contrast to these, we will also cover a use case corresponding to an open issue for ML research: adapting the recent developments in variational inference to anomaly detection (Sec. 6).

2. Data Quality Monitoring for the LHC Experiments

In this section, we overview the DQM scrutiny in the LHC experiments. We focus on the CMS experiment as we are most familiar with it. For showing a full picture, Sec. 2.5 presents an out-of-experiment example of monitoring the accelerator complex.

2.1. Overview

The LHC physics analyzes are performed only on *good-quality* data coming from the LHC collisions. Hence, prompt and accurate identification and flagging of the problematic data is required. In the CMS collaboration, imposing quality criteria is performed by the two main domains of the monitoring chain.

- *Online monitoring* provides live feedback on the quality of the data while they are being acquired, allowing the operator crew to react to unforeseen issues identified by the monitoring application.
- *Offline monitoring*, also referred to as *data certification*, was designed to certify the quality of the data collected and stored on disk using centralized processing (referred to as the event reconstruction, that converts detector hits into a list of detected particles, each associated with energy and direction).

The first online step is a prerequisite to an offline phase, in which detector experts monitor the data collected in a given period (typically a week) and decide which portion of the collected dataset meets the acceptance criteria. However, the two validation steps differ in three main aspects.

- The latency of the evaluation process. Online monitoring is required to identify anomalies in quasi-real-time to allow the operators to intervene promptly while the offline procedure has a typical timescale of several days.
- The fraction of the data which they have access to. Generally, CMS online processing runs at a rate of 100 Hz, corresponding to approximately 10% of the data written to disk for analysis (in order not to flood the monitoring system). The offline processing takes as input the full set of events accepted by the trigger system (~ 1 kHz of collisions).
- The granularity of the monitored detector components. While offline monitoring requires identifying the only overall status of the sub-detectors, online should determine faulty sub-detector elements.

Despite their specific characteristics, these two steps rely on the same failure detection strategy: the scrutiny of a long list of predefined statistical tests, selected to detect a set of possible known failure modes. The results of these tests are presented as a set of multidimensional *histograms* (mostly one-dimensional) for experts' convenience. The experts compare each distribution to a corresponding reference, derived from good-quality data in line with predetermined

validation guidelines. The good-quality data comes from the periods of the detector operating without any problems. The experts also look for unexpected effects that could affect analysis level quantities, e.g. noise spikes, dead areas or detector problematic calibrations.

2.2. *Experiment online legacy methods:*

The example of CMS drift tubes

The CMS failure detection algorithms focus on the interpretation of detector data organized in the form of histograms. The CMS DQM visualization tool, described in [6], displays those histograms organized geographically. The anomaly detection performed by the experts is very often related to identifying and discriminating healthy patterns from problematic ones. If such regions appear during the detector operation, the collaboration needs to know precisely when the problem appeared and how to intervene. Detector experts input their knowledge of the detector into binary classification algorithms targeting common and foreseen failure scenarios.

A class of these problems is based on counting the number of electronic hits per read-out channel. A concrete example could be the data recorded by the CMS drift tube (DT) chambers of the muon spectrometer, outlined in Fig. 1. It is an excellent illustration of an approach widely used by the CMS sub-detector communities and is referred to as *occupancy monitoring*.

The DT occupancy matrix can be viewed as a varying size two-dimensional array organized with a layer (row) and channel (column) indices. The method used in the online monitoring production system targets a specific failure scenario, by far the most frequent: a region of cells not providing any electronic signal, large enough to affect the track reconstruction in the chamber. It is usually related to temporary problems in the readout electronics. Examples of this kind of failures are shown in Figs. 1(b) and 1(c). The legacy strategy simply counts the area of dead (yielding exactly zero hits) regions without considering spatial proximity information. The strategy evaluates samples for each one of 250 DT chambers and assembles them in so-called *summary plots*. In this manner, human shifter, i.e. a

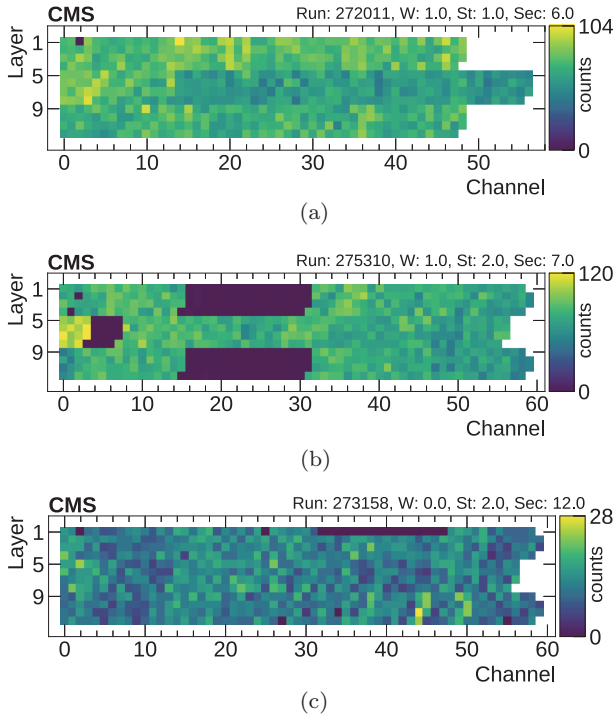


Fig. 1. Example of visualization of occupancy data for three CMS DT chambers. The data in (a) manifest the expected behavior despite having a dead channel in layer 1. The chamber in the plot in (b) instead shows regions of low occupancy across the 12 layers and should be classified as faulty. Figure (c) suffers from a region in layer 1 with lower efficiency, which should be identified as anomalous. From [1].

trained expert monitoring plots in real time, has a broad overview of the sub-detector status in *one* or a *few* plots. The first response human decision is based on the summary plot but the plot information is determined by an algorithm, such as the one described above, subject to performance fluctuations due to, for example, changing running conditions. For instance, the DT legacy occupancy monitoring strategy regards Fig. 1 instance (a) as non-problematic, correctly classifies the chamber in Fig. 1(b) as anomalous, but it is not sensitive enough to flag the chamber in Fig. 1(c).

The current level of automation extends to the infrastructure that creates the plots and the superposition to the existing reference. For

some sub-detectors, a statistical test (e.g. Kolmogorov–Smirnov, χ^2) is performed, but the interpretation and ultimate decision are again taken by the human shifter.

2.3. *Legacy trigger rate monitoring*

A further category of online monitoring is *trigger rate monitoring*. The *trigger system* is an essential part of the LHC acquisition process and the start of the physics event selection process. The LHC operates at the remarkable collision rate of 40 MHz and each event corresponds up to several MBs of data in unprocessed form. Due to understandable storage constraints and technological limitations, each experiment is required to reduce the number of recorded data.

At CMS, a hierarchical set of trigger algorithms [7] are designed to reduce the event rate while preserving the physics reach of the experiment. The CMS trigger system is structured in two stages using an increasingly complex information and more refined algorithms. The Level 1 (L1) Trigger is implemented on custom electronics and reduces the 40 MHz input to a 100 kHz rate. High-level trigger (HLT) is a collision reconstruction software running on a computer farm, which scales the 100 kHz rate output of L1 Trigger down to 1 kHz. The HLT nodes (or *paths*) are seeded by the events selected by a set of L1 Trigger outputs.

The event acceptance rate is affected in the presence of several issues, e.g. detector malfunctions. Depending on the nature of the problem, the rate associated with specific paths could change to unacceptable levels. In such cases, the system should alert the shift crew, calling for problem diagnosis and intervention. Critical cases include dropping to zero or increasing to extreme values.

The rate of the physics processes determining the trigger rate decreases with the luminosity and, as a consequence, with pile-up (PU), a number of proton–proton collisions in the same event. Consequently, the recorded collision rates decrease as well as they primarily depend on the luminosity of the beams. In practice, trigger monitoring predicts an average rate per bunch-crossing as a function of an average measurement of the PU for each period. These predictions are then compared to the recorded rates as data are being collected,

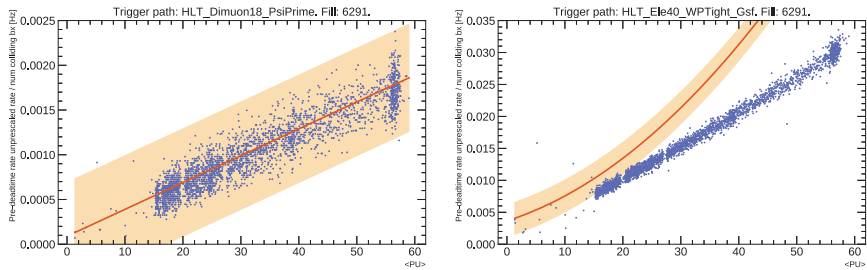


Fig. 2. Observed trigger rates as a function of average PU (blue dots), compared to the predicted dependence (red line) and its uncertainty (in the orange band) generated using the monitoring software. The plots above show an example of a well (left) and poorly (right) predicting model. From [8].

spotting small and unexpected deviations. In Fig. 2, the red lines correspond to the predictions, while the blue dots are the actual values readout. The model describing the expectation is derived from a best-fit approximation (i.e. fitting the rate values as a function of average PU) limited to linear, quadratic or exponential regression. The prediction models are generated ahead of time using recent, good-quality data. The final regression model is selected based on least-squares minimization, with a bias towards more straightforward (i.e. linear) fits; each trigger node is fitted independently from others. The models are updated periodically (approximately every other month) to account for changes, e.g. in the sub-detectors, trigger algorithms or calibration updates.

2.4. Experiment offline data certification

The data certification step performs routine physics level checks on physics objects, i.e. hadrons, leptons, photons, and so forth when experts look for anomalies in the statistical distributions of fundamental physics quantities. In CMS collaboration, the monitoring is based on histograms produced during the offline data reprocessing. The outcome of this task is the classification of collected datasets into data usable for physics analysis (good data) and data to be discarded (bad data). A finer granularity is also possible but we will not enter in the details here.

The collision data are collected as a series of time blocks. In CMS, these blocks are called luminosity sections (LS), corresponding to approximately 23 s of consecutive data. The LS is indivisible and if something goes wrong in a given LS, the full block is rejected.

Since the offline reconstruction is more accurate than what is available online, the data certification can be more effective in spotting problems. Now, at the CMS experiment, the procedure is completely human-based.

2.5. Accelerator monitoring example with sensor data

Besides relying on physics data, the sensor (non-collision) data is commonly used for monitoring the complex apparatuses in other aspects of HEP, e.g. the detector magnets, the detector gas systems, cryogenics. Apart from monitoring the experiments, the CERN LHC accelerator complex needs dedicated monitoring of the accelerators itself [9]. In this subsection, we will overview one such application.

A critical component of the LHC is its superconducting magnets which store a substantial amount of magnetic energy. Consequently, the cables responsible for powering the system conduct the current at the level of 12 kA in the magnetic field of 8.5 T. Those superconducting cables are not *cryostable* thus a random and local temperature change can lead to a sudden transition to a normal conduction state [10], known as a *quench*. During operation, the temperature can locally elevate above a critical value and lead to cable damage. Quenches may occur in various circumstances but some of the most common ones take place during a so-called magnet training. At the first powering during ramping up a current, magnet loses superconducting state long before reaching the expected critical current. At the next attempt of powering, the current that could be reached before quench is higher. The process continues during succeeding attempts, and the maximum current that could be reached increases quench after the quench, slowly approaching a plateau.

Since most of the high-current superconducting magnets used in the LHC are not self-protected the quench protection system (QPS)

was introduced [11, 12]. This system consists of a quench detection system (QDS) and actuators which are activated once a quench is detected. A superconducting magnet has zero resistance and a relatively large inductance. When a constant current flows through the magnet, the total voltage across it is zero. With quench the resistance becomes non-zero, hence, a voltage develops over the resistive part. The QPS uses the measured voltage to detect the quench. However, during normal operation the inductive voltage may be above the resistive voltage detection threshold and thus must be compensated to prevent the QDS from spurious triggering. The most important part of the quench detector is an electronic module for extracting the resistive part of the total voltage. The triggers are transmitted to other protection devices via current loops to initiate a safe shut-down of the electric circuits supplying the magnets.

A quench candidate is validated as a real quench or noise by a timing discriminator. The alarm is raised when the voltage resistive component is higher than a threshold for the time interval longer than a validation time. A desirable extension to the current implementation is a system modeling and predicting voltage readouts allowing for faster detection and prevention of quench events.

3. Machine Learning Anomaly Detection for HEP DQM

Anomaly detection is one of the oldest problems of statistics. Accordingly, the most principled approach to anomaly detection is density estimation. However, simple parametric univariate density estimation of the normal behavior is doomed to failure in moderate to high dimension [13]. ML anomaly detection has become the standard alternative in this case. In very broad terms, the ML anomaly detection addresses the dimensionality issue with three approaches of increasing complexity: learning a decision function, which is much simpler than full density estimation; learning a representation, which projects (usually non linearly) the data in a more convenient space, and finally tackling the dimensionality issues of parametric density estimation with variational methods.

These three approaches have been hybridized with neural networks exploited for their capacity of universal function approximators. As a consequence, the ML methods of anomaly detection have significantly changed in the last years. While a relatively recent general survey on anomaly detection like [14] describes a wide variety of specific methods, the present trend is to adapt general-purpose neural network based systems, such as the various flavors of deep neural networks (DNN), autoencoders and generative models [15], to anomaly detection. This chapter illustrates the benefits of this trend. Generally speaking, for our case studies, neural network-based solutions provide satisfactory results when compared to pre-deep learning reference methods.

HEP DQM offers very interesting tests for the applicability of these new trends to real-world data. In HEP DQM, the data always exhibit significant dimensionality, making the problems non-trivial. Also, the operational requirements are high: on computational efficiency, given the vast volume of data to monitor; on performance, given the fact that the data are generally noisy; finally, but most importantly, the solution must be usable in a production system, which implies simplicity, for implementation and debugging purpose, as well as credibility, supported by some level of interpretability.

An essential question is which type of learning is made possible by the data. Anomaly detection implies the lack of a complete set of representative examples of all possible behaviors. If such representative examples are available, anomaly detection reduces to *binary classification* (supervised learning). *Semi-supervised* anomaly detection assumes the availability of both examples of the regular behavior and unlabeled ones; *unsupervised* anomaly detection assumes no labels at all. *Unitary* (or *one-class* learning) is the case where only examples of the regular behavior are exploited at training time.

Fully unsupervised approaches based on the neighborhood (e.g. distance based outlier analysis), topological density estimation (e.g. Local Outlier Factor and its variants), or clustering [16] miss at least one of the listed requirements. These methods have quadratic complexity. Moreover, they poorly perform in high dimensions because of the *curse of dimensionality* [17]: in high dimensions, all pairs of

points become almost equidistant [18, 19]. But the most important issue is that a simple geometric distance in the feature space does not define a useful similarity metric in our case.

Learning a decision function through binary classification is a valid option for DQM when specific anomalous scenarios have been extensively studied. While training times might be long, the inference is usually fast. Typically, the experiments keep copious archives of subdetector-specific quality-related quantities, e.g. the CMS DT occupancy plots. Convolutional neural networks (CNNs) [20] are a natural choice for image-like inputs, as they integrate the basic knowledge of the topological structure of the input dimensions and learn the optimal filters that minimize the objective error.

However, there are good motivations for the unitary (one-class) approach. Mainly, the fact that examples and/or labels of anomalies may not be natively available. In the next sections, we will show examples where requesting expert labeling makes sense or not. The pre-deep learning reference methods for this approach are μ -SVM [21] and isolation forest [22, 23].

Deep architectures have become increasingly popular in semi-supervised anomaly detection [15]. They cope with the issues of conventional methods discussed in the previous paragraphs. The need for agnostically learning a representation from the data can be addressed indirectly by DNNs in a classification or regression context [24], and can be exploited for semi-supervised anomaly detection [25]. However, the information related to anomaly can be lost if it is not relevant for the specific task they address. The better alternative is learning a direct encoding, with an autoencoder. DNN-based autoencoders [26] are parametric maps from inputs to their representations and are trained to perform an approximate identity mapping between their input and output layers. The network maps an input to a usually low-dimensional representation. Autoencoders are particularly suitable to anomaly detection: when trained on the good-quality samples, unseen faulty samples tend to yield sub-optimal latent representations and, as a consequence, decoder outputs, indicating that a sample is likely generated by a different process. Furthermore, the encoded representation space may distinguish the anomalous regions alone.

Until relatively recently, the autoencoding approach was restricted to learning a deterministic map of the inputs to the representation, because the inference step with probabilistic representations would suffer from high computational cost [27]. A considerable body of work has been devoted to regularize the deterministic architectures, implicitly learning a density model [28].

The dissemination of the generative models, and specifically the Variational Autoencoder (VAE) [29, 30], offers a more general and proper path where the learned representation is the variational approximation to the posterior distribution of the latent variables given an observed input. A straightforward approach for VAE-based anomaly detection [31] considers a simple VAE and the Monte-Carlo estimate of the expected reconstruction error. However, [32] discusses two possible intrinsic limitations. Firstly, because the model is trained only on inliers, the representation will not be discriminative, and will essentially overfit the normal distribution; besides, the representation might even be useless, falling back to the prior [33, 34]. On other grounds, the general ability of deep generative architectures to point anomalies using the model likelihood has been questioned [35, 36].

Reference [37] address the former issues with specific hypotheses on the distributions of inliers and anomalies. A more general approach [32, 38] exposes the model to out-of-distribution examples, without knowledge of the actual anomaly distribution, with adversarial architectures and ad-hoc regularizations. Overall, neither of these approaches would meet the robustness and simplicity specifications of our motivating application. In Sec. 6, we show that a VAE exploiting the natural conditional structure of the problem and trained with a regular loss function is effective for anomaly detection in the context of the trigger system monitoring.

4. Detector Components Anomaly Detection with Convolutional Neural Networks and Autoencoders

In this section, we highlight the results presented in [1]. The goal of the study is to detect anomalies in the CMS muon spectrometer based on occupancy plots (see Sec. 2.2). CNNs and convolutional

autoencoders were used. The approach consists of three complementary strategies, as summarized in the following table:

Name	Motivation	Input	Type	Method
local	replace	layers	supervised	CNN
regional	extend	chambers	one-class	Autoencoder
global	extend	chambers	unsupervised	Autoencoder

4.1. *Supervised anomaly detection*

The problem was first approached as a supervised image classification task, as the plots from Fig. 1 can be interpreted as such. Moreover, the imbalance between good and bad data is not extreme. The anomalies are then frequent enough for a sizable set of them to be used for binary classification. The *local* method exploits the geographical information of the detector assessing the misbehavior with the highest reasonable granularity and then combining the results to probe different detector components.

The problem falls into a list of known and frequent issues with the readout electronics. To solve it, the data was labeled by detector experts. The ground truth was established on a random subset of the dataset, by visually inspecting the input sample before any preprocessing: 5668 layers were labeled as good and 612 as bad. Subsequently, the input was preprocessed by fixing the input dimension, smoothing and normalization. The 9.75% fault rate is a faithful representation of the real problem at hand. Out of this set 1134 good and 123 bad examples were reserved for composing the test set corresponding to 20% of the labeled layers. The input dimension (i.e. the number of features) was low, allowing for comparison between various algorithms, including the ones sensitive to the number of features, as discussed in Sec. 3. The following methods were compared:

- **unsupervised learning** with a simple statistical indicator, the variance within the layer, and an image processing technique, namely the maximum value of the vector obtained by applying a variant of an edge detection Sobel filter;
- **one-class learning**, with isolation forest, and μ -SVM;

- **supervised learning**, with a fully connected shallow neural network, and a CNNs.

CNN was chosen because the problem at hand is naturally linked to image processing. In contrast to other traditional algorithms that may overlook spatial information between pixels, CNN effectively uses adjacent pixel information to extract relevant variability in data using learned filters and use a classification layer at the end. The shallow neural network model matched the number of parameters in the CNN to obtain a term of comparison for the CNN.

The architecture of the CNN model with one-dimensional convolution layers used in this study is shown in Fig. 3. As the number of training samples was low, the architecture had to leverage this with a small number of trainable parameters to limit over-fitting. Rectified linear units were chosen as activation functions for inner-layer nodes, while the softmax function is used for the output nodes.

The performance of the various models on a held-out test data set is shown in Fig. 4. The supervised deep learning outperforms

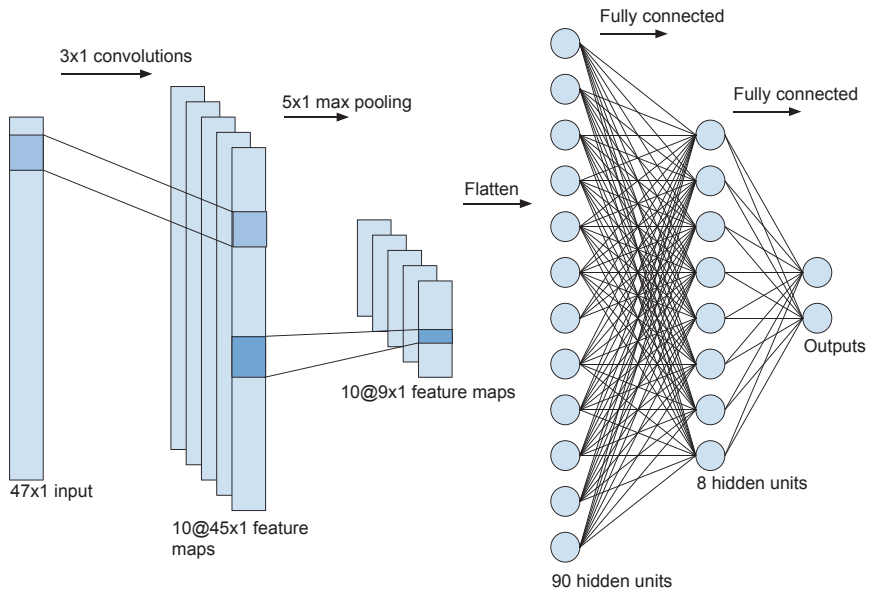


Fig. 3. Architecture of the CNN model used to target the CMS DT occupancy monitoring. From [1].

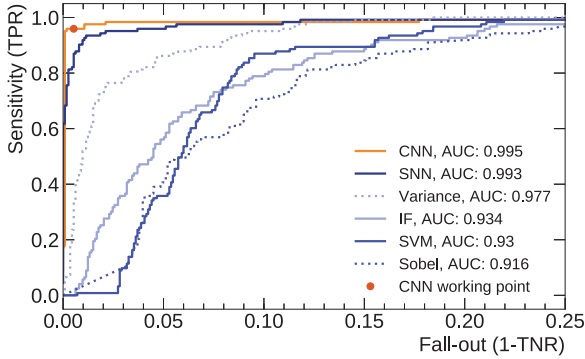


Fig. 4. ROC curves for different models used in the local approach. From [1].

the other methods. Thanks to the limited number of parameters of the model, the training converges to a satisfactory result, even though the number of training samples was small. Although the area under the curve (AUC) of the fully-connected SNN is comparable to the one of CNN, the initial edge detection filter provides marginal improvement. The edge detection filters to learn were not simple contrasts, as shown by the poor results of the Sobel filter method. The limited performance of the isolation forest is likely to come from the violation of its fundamental assumptions: the anomalies should be rare, and isolated in the native feature space. The faults were not rare (fault rate approaching 10%) and homogeneous. The inferior performance of the typical one-class method μ -SVM illustrates the well-known smoothness vs. locality argument for deep learning: the difficulty in modeling the highly varying decision surfaces produced by complex dependencies involving many factors. For μ -SVM, the implicit prior of kernel-based classification is that the function to be learned is smooth such that generalization can be achieved by local interpolation between neighboring training examples. As argued at length by [27], this assumption is questionable for high data dimensionality. Moreover, all baseline methods lose a piece of critical information: the local geometric relationship in the data related to the underlying apparatus.

The legacy strategy produces a chamber-wise goodness assessment without being capable of identifying a specific problematic layer

in the chamber. For this reason, a direct comparison with the CNN model is impossible. The loose estimate (based on returned problem severity) estimated the specificity of the legacy strategy to 91%, with a sensitivity of only 26%.

As discussed in Sec. 3, the interpretability of the results is one of the requirements in the HEP domain. Unfortunately, the CNN filter visualization did not provide vital information for the human experts as the nature of the data is much different than the real-world data sets. However, the CNNs decision can be understood through saliency maps [39]. Example of such visualization generated for DT occupancy plots is shown in Fig. 5. The channels with high values match the anomalous regions. These plots were proven fundamental to point the detector experts to the root of the CNN decision allowing them to carry on further investigations on the detector aspects. In

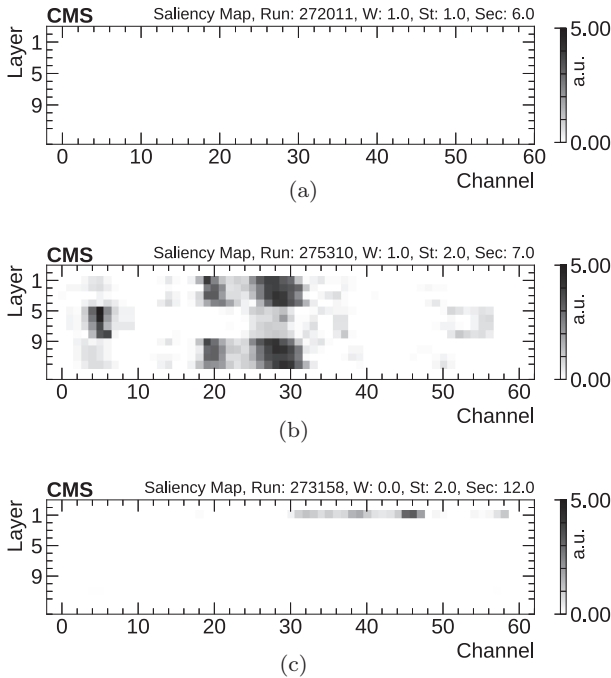


Fig. 5. Example of visualization of saliency maps for three CMS DT chambers corresponding to input occupancy plots from Fig. 1. The scale is proportional to the channel influence over classifier decision to flag problems. From [8].

the future, in case of incorrect classification, the saliency maps could be used to understand the decision of the algorithms in detail and to take corrective measures.

The CNN model has been integrated into the CMS DQM infrastructure at the beginning of the 2018 LHC Run and kept running in parallel with the legacy strategy. That allowed to commission it using the newly acquired collision data. After initial tuning of the working points to meet the requirements of the DT detector experts, the algorithm has been performing reliably, and it is considered for deployment in the next LHC Run.

4.2. One-class anomaly detection

In normal conditions, the healthy DT chambers show similar occupancy levels in adjacent layers with the four inner layers having a different behavior due to their different spatial orientation. The convolutional autoencoder used in [1] exploited the patterns of relative occupancy of the layers within a chamber. This approach extended and complemented the one presented Sec. 4.1, allowing to identify less frequent intra-chamber problems which require the comparison of the information about all layers within one chamber to be spotted. Typical examples of these kinds of failures are problems related to the high-voltage bias. The voltage distribution system is organized by layers and a lower value with respect to the nominal operation point would result in lower detector efficiency and, as a consequence, lower absolute occupancy in the affected region.

In this study, the dataset was cleaned from the common anomalies using the CNN model from Sec. 4.1 to save time on manual labeling and acquire sizeable dataset. Then the autoencoder-based model was trained to properly reconstruct healthy behavior. Finally, the autoencoder-based model was tested on a set of occupancy plots where chambers showed problems in a particular layer (layer 9). Figure 6 shows that the mean squared reconstruction error (MSE) integrated over healthy regions (even from anomalous chambers) is lower than the one from anomalous ones. Of course, the severity of the problem matters and layers operating in 3450 V are more difficult to be detected than the ones operating in 3200 V. To summarize, the

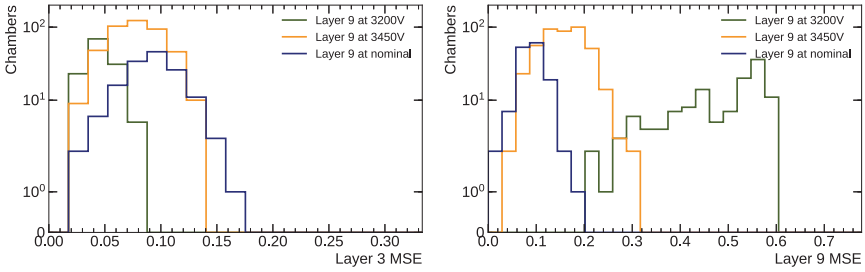


Fig. 6. MSE between reconstructed and input samples for layer 3 (left) and layer 9 (right) for three categories of data. Despite a problem in layer 9, all MSEs for layer 3 are comparable for all chambers. The nominal voltage falls between 3550 and 3600 V. From [1].

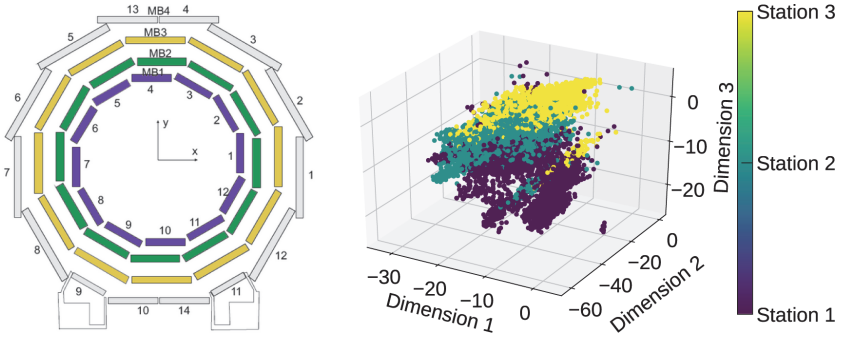


Fig. 7. Compressed representations of chamber level data for all chambers in the dataset. The representations are clustering (right) according to their positions in the detector, i.e. station number. The DT numbering schema is shown on the left. From [1].

detector experts can design custom metrics, integrating the MSE over the areas of interest to further extend the monitoring infrastructure.

4.3. Unsupervised anomaly detection

Finally, Pol *et al.* [1] showed that a byproduct of the undercomplete autoencoders, i.e the lower dimensional latent representation, can be further plotted to visually track novel behavior patterns and emerging problems. For instance the DT chambers three-dimensional latent representation clusters according to the chamber position in the CMS detector, shown in Fig. 7. When the chamber behavior

changes, so will the manifold. Higher dimension manifolds can be used as well, e.g. to perform classification.

5. Data Certification Novelty Detection with Deep Autoencoders

This section presents an approach of applying autoencoders to automate the DQM scrutiny, with the example of the CMS data certification process (Sec. 2.4) and results from [2]. With a tolerance for false negatives, the autoencoders will reduce the manual work as discussed in [40]. Pol *et al.* [2] used data for the physics certification process (see Sec. 2.4). The dataset used in this work consisted of 163 684 LSs recorded from June to October 2016. In total, 401 physics variables were used (e.g. transverse momentum, energy, multiplicity, direction for the different physics objects). The binary quality labels determined by the manual certification procedure performed by the detector experts were used for evaluating model's performance.

The human experts make decisions regarding the data quality based on the shape of the statistical distributions of key quantities represented in the form of histograms. In the case of an anomaly, the corresponding histograms should show a considerable deviation from the nominal shape (for visual interpretation see Fig. 8). To mimic

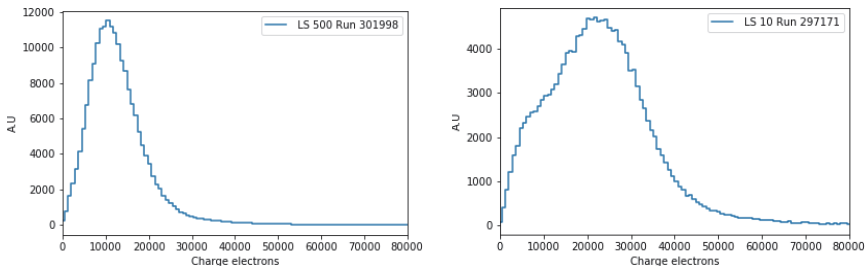


Fig. 8. Two examples of histograms related to the CMS pixel detector status for a normal (left) and anomalous (right) LS. The reference shape is the Landau distribution. The bad LS manifests anomaly in low charge, which is caused by the pixel detector not being properly synchronized with the bunch crossing. Such distributions, obtained for each LS, are preprocessed into a summary statistics vector of seven variables: five quantiles, mean and standard deviation. From [8].

this logic, the distribution $D_i = \{x_0, \dots, x_k\}$ of each one of the 401 used variables was represented by its summary statistics using five quantiles, mean and standard deviation. The final vector has 2807 features. Each data-point represents the data acquired during one LS to aim for high time granularity of the classification results. The high dimensionality and nonlinear dependencies between variables preclude the use of traditional anomaly detection techniques. Instead, different autoencoder regularization techniques were examined. The final receiver operating characteristic (ROC) curves of autoencoders and their corresponding AUC are shown in Fig. 9.

Beyond performance, a valuable model for the certification task needs to provide easily interpretable results allowing the experts to pinpoint the root of a problem. In this respect, the autoencoder approach provides a clear advantage allowing to evaluate the contribution to the MSE metric of each input variable. Misbehaving variables can be easily singled out based on their high contribution to the overall error. Figure 10 illustrates one example of how this can be exploited on the CMS data. The features are grouped according to their sensitivity to a particular physics property. The plot of the absolute error allows the expert to identify the problematic area at a glance judging on the absolute size of the error for the variable or group of variables.

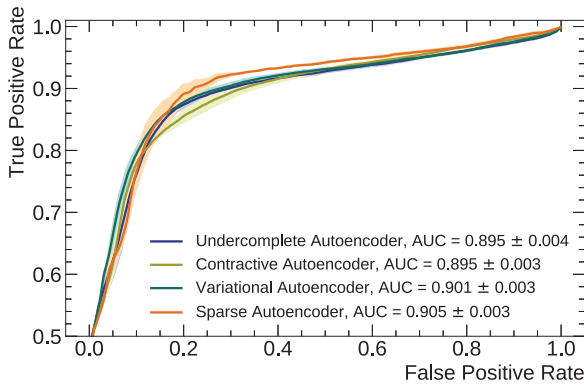


Fig. 9. ROC and AUC of the autoencoder models using different regularization techniques. The bands correspond to variance computed after running the experiment five times using random weight initialization. From [2].

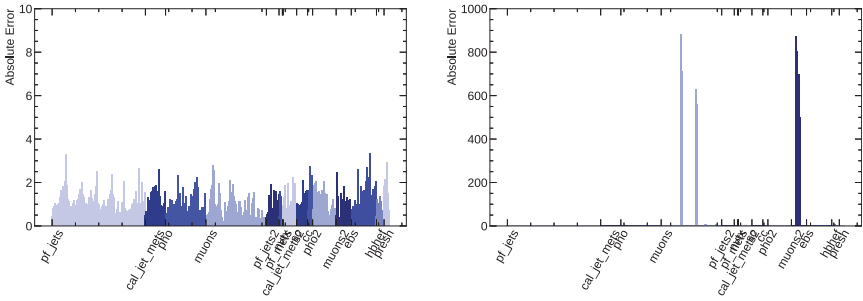


Fig. 10. Reconstruction error of each feature for two samples. Different colors represent features linked to different physics objects. For a negative sample (left) similar autoencoder reconstruction errors are expected across all objects with small absolute scale. Anomalous samples (right) have visible peaks for problematic features (muons). From [2].

6. Trigger Rate Anomaly Detection with Conditional Variational Autoencoders

Reference [41] targets improving anomaly detection for the trigger system (Sec. 2.3) with VAEs. To avoid the pitfalls described in Sec. 3, the key is to exploit the hierarchical structure of the trigger system to input all available observation into the VAE, to constrain the representation, by separating the known factors from the other unknown sources or variability. The model, called AD-CVAE, includes the architecture, as a specific realization of conditional variational autoencoders (CVAE) [42–44], as well as the corresponding loss function and a detection metrics. Overall, the contribution shows that a regular CVAE architecture can be exploited for general anomaly detection tasks in HEP context. More details and experiments are available in [41].

6.1. Problem statement

The current CMS trigger monitoring system is based on the comparison between the observed per-node rate and its reference value for the measured PU value. While the current implementation is quite effective in spotting erratic changes for a single node, it is less sensitive to collective changes on several nodes that could equally affect

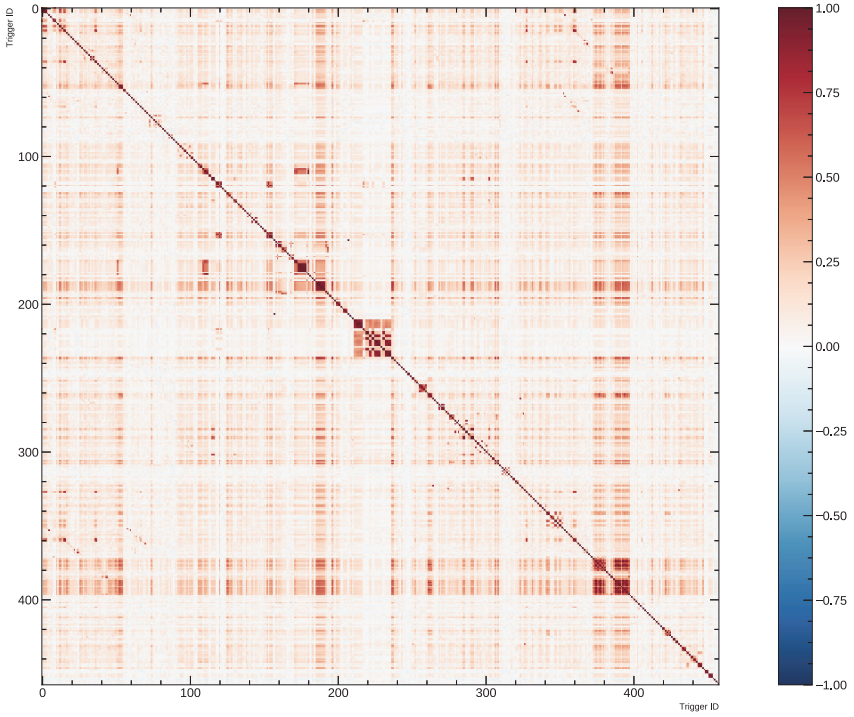


Fig. 11. Correlations between 458 HLT rates of fill 6291 of LHC Run 2. From [8].

the overall acceptance rate. In particular, about 600 nodes of the HLT can be grouped in several *configuration groups*, showing strong correlations in their acceptance rate variations, see Fig. 11.

The dominant cause of correlation is structural, known and measurable: the direct, pre-configured link from a set of L1 nodes to an HLT node through a specific configuration (Fig. 12). However, more subtle and un-reported causes can create correlations: physics processes when different nodes select the same physics objects with different requirements (e.g. different requests on its energy); or utilization of the same sub-detector component or software component across different nodes. The corresponding graphical structure must include these unknowns.

To correctly model the trigger system, the algorithm has to successfully disentangle the dependence of HLT rates on L1 rates from

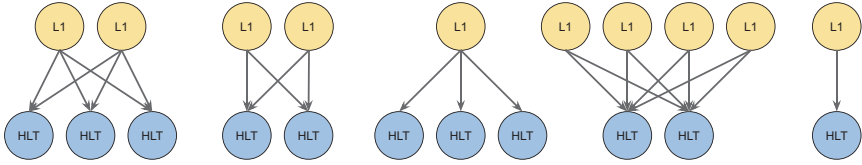


Fig. 12. Simplified, schematic graph inspired by the trigger system configuration. Blue nodes represent HLT while yellow L1. Each link is unidirectional starting from yellow nodes. For each LHC fill, the graph has a few hundred nodes. The connection between L1 and HLT nodes can be seen as a hierarchical directed graph from the L1 to the HLT system. From [41].

all other unknown processes. In light of the results of [45], the disentanglement objective in generative models cannot be met by fully unsupervised VAE architectures. The alternative is to enforce disentanglement through a *structured conditional architecture*.

The second issue is: what are the anomalies, and the normal behavior? We are interested in highlighting instances where we observe:

- big change on a single feature called **Type A** anomaly (to reproduce the functionality of the current monitoring), *or*
- small but systematic change in a structural configuration group, called **Type B** anomaly (novel strategy).

On the contrary, an instance x with a problem of small severity and on a group of uncorrelated features should be considered as an inlier, corresponding to expected statistical fluctuations.

While dealing with Type A anomalies is relatively well managed, the CMS experiment currently does not provide any tools to track problems falling into the Type B category.

6.2. The architecture

The goal of the architecture is to address the disentanglement issue, in other words to build a representation within the VAE framework where the known and unknown factors are identified. This includes both the structure of the representation, and a loss function that takes into account the conditioning on the known factors. The formal

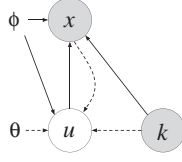


Fig. 13. An example of CVAE as a directed graph. Solid lines denote the generative model $p_\theta(x|u, k)p_\theta(u)$. Dashed lines denote variational approximation $q_\phi(u|x, k)$. Both variational parameters θ and generative parameters ϕ are learned jointly. From [41].

model is as follows: the observable x is a function of k (known) and u (unknown) latent vectors, i.e. $x = f(k; u)$. k and u are assumed to be marginally independent. In the trigger context, x is the feature vector of observed HLT rates $[x_1, x_2, \dots, x_n]$, k is the vector of observed L1 rates, and u stands for the unknown factors. Conceptually, features associated with the same subset of the k vector correspond to a *structural configuration group*. The variable u allows for modeling multiple modes in the conditional distribution $p(x|k)$ making the model sufficient for modelling one-to-many mapping.

This defines the conditional directed graphical model of Fig. 13, where the input observations modulate the prior on latent variables to model the distribution of high-dimensional output space as a generative model conditioned on the input observation. The conditional likelihood function $p_\theta(x|u, k)$ is formed by a nonlinear transformation, with parameters θ . ϕ is another nonlinear function that approximates inference posterior $q_\phi(u|k, x) = N(\mu, \sigma I)$.

The ϕ and θ functions are implemented as deep neural networks with nonlinear activation functions. Figure 14 shows the autoencoder architecture corresponding to Fig. 13 as a block diagram.

This model, called AD-CVAE, is trained efficiently in the framework of stochastic gradient variational Bayes. The usual loss function of VAE is the so-called evidence lower bound, which is a tractable proxy for optimizing the log-likelihood of the data. With the conditioning on k taken into account, the modified objective lower bound is

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x, k)}[\log p_\theta(x|z)p_\theta(x|k)] - \mathbb{D}_{\text{KL}}(q_\phi(z|x, k)||p(z)), \quad (1)$$

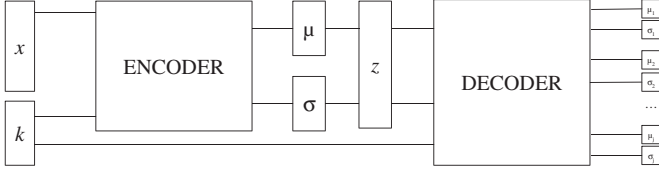


Fig. 14. Architecture of CVAE targeting trigger system anomaly detection. Observable data x depends on z (capturing non-observable factors of variation u) and k vectors. From [8].

where z (Gaussian latent variable) intends to capture non-observable factors of variation u .

6.3. The loss function

The original works on VAEs by [29, 30] proposed a full (diagonal) Gaussian observation model, that is

$$P_{\theta}(x|z) = \mathcal{N}(\mu, \sigma I),$$

where both the multidimensional mean vector and the multidimensional variance vector are to be learnt. However, in most practical applications the VAE evaluates the reconstruction loss with a simple mean squared error (MSE) between the data x and the output of the decoder. Such an approach suffers from a very serious issue. It is equivalent to setting the observation model $p_{\theta}(x|z)$ as a normal distribution of fixed variance $\sigma = 1$.

Fixing the variance, this way can be detrimental to learning as it puts a limit on the accessible resolution for the decoder. Instead, the model can learn the variance of the output of the decoder feature-wise (i running as the dimensionality of the data vectors x):

$$-\log p_{\theta}(x|z) = \sum_i \frac{(x_i - \mu_i)^2}{2\sigma_i^2} + \log \left(\sqrt{2\pi} \sigma_i \right). \quad (2)$$

Learning the reconstruction variance allows the model to find the *optimal reconstruction resolution* for each feature of the data, separating the intrinsic noise from the actual data structure. Although it has been argued that this approach can challenge the optimization

process [33, 46], there were no reported challenges when training the AD-CVAE.

After inserting Eq. (2) as the reconstruction objective to the general loss defined in Eq. (1), the final objective of AD-CVAE is

$$\begin{aligned} \mathcal{L}_{\text{AD-CVAE}}(x, k, \theta, \phi) \\ = \sum_i \frac{(x_i - \mu_i)^2}{2\sigma_i^2} + \log \left(\sqrt{2\pi}\sigma_i \right) + \mathbb{D}_{\text{KL}}(q_\phi(z|x, k)||p(z)). \end{aligned} \quad (3)$$

6.4. Anomaly metrics

Once the model parameters are learned, one can detect anomalies:

- of type A with average infinity norm of the reconstruction loss $m_A = \|\frac{1}{\sigma}(x - \hat{x})^2\|_\infty$, where \hat{x} is the reconstructed mean and σ is the reconstructed variance of decoder output;
- of type B with KL divergence $m_B = \mathbb{D}_{\text{KL}}(q_\phi(z|x, k)||p(z))$, known as information gain.

In the first case, an anomaly is identified on a single feature. For a given data point (x, k) , the evaluation of the loss of the VAE at this data point $\mathcal{L}(x, k)$ is an upper-bound approximation of $-\log p_\theta(x|k)$, measuring how unlikely the observation x is to the model given k . AD-CVAE thus provides here a model that naturally estimates how anomalous x is given k , rather than how anomalous the couple (x, k) is. It means that a rare value of k associated with a proper value for x should be treated as non-anomalous, which is the goal. The binary indicator is obtained by thresholding the value, a typical strategy for anomaly detection. With thresholding, the choice of the infinity norm of the reconstruction error instead of the mean is required. A mean of the reconstruction error would be uninformative when most of the features do not manifest abnormalities and, as a consequence, lower overall anomaly score.

As argued in [47], the \mathbb{D}_{KL} measures the amount of additional information needed to represent the posterior distribution given the prior over the latent variable being explored to explain the current observation. The lower the absolute value of \mathbb{D}_{KL} , the more

predictable state is observed. The \mathbb{D}_{KL} was then used as a *surprise* quantifier, e.g. in [47, 48] when the model was exposed to held-out images. Nalisnick *et al.* [35] and Snoek *et al.* [36] explored \mathbb{D}_{KL} as an indicator of out of distribution samples. For type B outliers, the expected anomaly systematically reinforces patterns in data. It is then expected that not calibrated model allocates such information using the latent bits, allowing for a successful reconstruction. On the other hand, changes in uncorrelated features will be removed in the encoding process, resulting in low-reconstruction likelihood. Hence anomalous input yields higher values of m_B and likelihood at the same time. Thus, m_B must be detached from the reconstruction part of the loss function as combining metrics is detrimental to the detection results.

Because of two separate failure scenarios, the metrics are not combined in one overall score but rather use logical **OR** to determine anomalous instances.

6.5. *Experimental results*

CVAE model was evaluated on two datasets: a synthetic one and on the real trigger dataset. The synthetic data set is a version of the Gaussian mixture model and was implemented as an initial benchmark that proxies the trigger data set. For testing, the samples are generated according to the following table:

Test set	Description
Type A Inlier	Generated in the same process as training data
Type A Anomaly	5σ change on ϵ for a random feature
Type B Inlier	3σ change on ϵ for a random set of correlated features
Type B Anomaly	3σ change on ϵ for a random feature cluster

The choice of 5σ and 3σ comes from the legacy requirements of our target application. The real HLT rates are treated as x and L1 Trigger rates as k . The proposed prototype used four L1 trigger paths

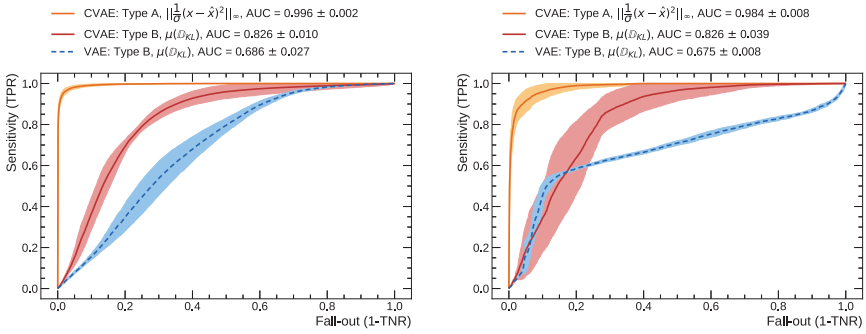


Fig. 15. The ROC curves for two anomaly detection problems using synthetic (left) and CMS trigger rates test dataset (right). The bands correspond to σ computed after running the experiment five times. From [41].

that seeded six unique HLT paths each. The dataset totaled 102895 samples from which 2800 samples were used for testing. Again the hypothetical situations that are likely to happen in the production environment were considered. Four synthetic test datasets were generated manipulating the test set similarly to the synthetic dataset (based on the table above).

The results are reported in Fig. 15. Given the high order of the deviation on Type A anomalies, the model easily spots them. Also, Type B detection results show that CVAE is outperforming VAE baseline and confirming it is suitable for a task in question. The performance of the algorithm on CMS dataset is matching the performance we reported for the synthetic one.

7. LHC Monitoring with LSTMs

Recurrent models can be applied to temporal or sequential data, where the order of data is important. Recurrent neural networks (RNNs) [49] can process sequential data element-by-element. In this way, they can model sequential and time dependencies on multiple scales. However, the influence of a given input on hidden and output layers during the training often results in gradient either decaying or exponentially grow as it moves across recurrent connections. This effect is described as the vanishing or exploding gradient problem.

A successful attempt to prevent this phenomenon is the long-short-term memory (LSTM) network [50], through the introduction of internal state node and forget gate.

In this section, we summarize the results of the experiments from [51]. The authors validated the performance of the LSTM network in a voltage time series modeling task, see the description of the problem in Sec. 2.5.

The data used in the experiments consisted of many years of magnet activity. A group of 600 A magnets, that generated the highest number of quench events, were used. The anomalous events were not only sparse but also challenging to find, as the logging database does not enable automated quench periods extraction. The authors developed an extraction application, automating the dataset generation. In the experiments, different lengths of time window frame were considered. Finally, the 24 h window ahead of a quench event were chosen, totaling 425 from the period of 2008 and 2016.

The LSTM model yielding the best results is shown in Fig. 16. The tests considered the ability of a model to anticipate forward

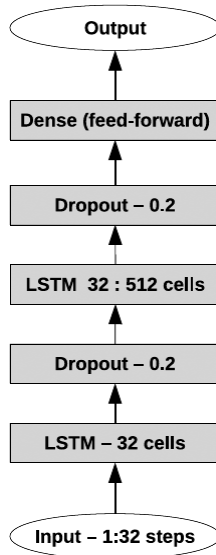


Fig. 16. The LSTM-based neural network used for the experiments in [51].

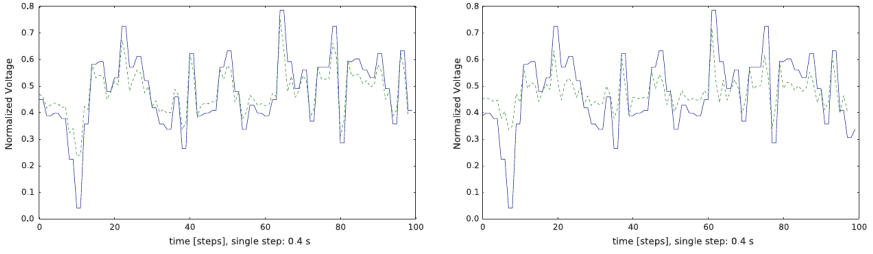


Fig. 17. The LSTM-based neural network voltage predictions for one step ahead (left) and two steps ahead (right). From [51].

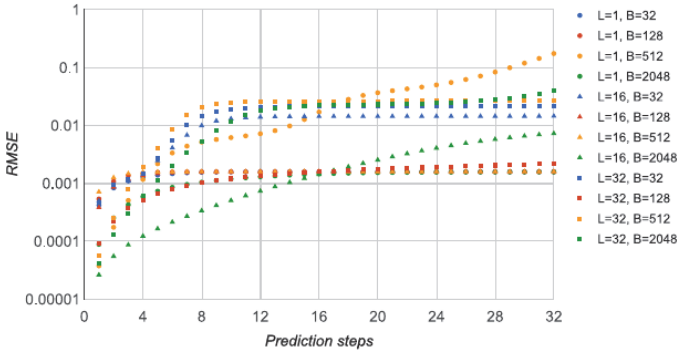


Fig. 18. The value of RMSE as a function of prediction steps for different batch size B and the number of previous time steps L values with 32 neurons in the middle LSTM layer. From [51]

voltage values. Figure 17 shows the predicted voltage readings. The authors used root mean square error (RMSE) and mean percentage error (MPE) to assess the algorithm performance. The RMSE results are presented in Fig. 18, where L corresponds to the number of previous time steps as input and B corresponds to a training batch size. The best results were obtained with $L = 16$ and $B = 2048$. After verifying that the model can predict forward voltage readings, the ultimate challenge was to select a threshold of RMSE value determining which readings should be considered anomalous. Unfortunately, this value has not been chosen and requires further investigation.

The resulting model promises to speed up the quench detection and prevention process. Besides the model, the authors developed

a visualization framework and tested the model on FPGAs, as the system reaction time is critical.

8. Conclusion

In this chapter, we discussed novel approaches to improve the accuracy of data quality applications for high-energy physics experiments. Taking as an example the CMS experiment at the CERN LHC, we showed how anomaly detection techniques based on machine learning algorithms could detect unforeseen detector malfunctioning. We also showed how the flexibility of deep learning architecture allows one to enforce known causal relation between data, through constraints built by connections in the network architecture. The results demonstrate that the techniques based on DNNs provide a breakthrough for complex and high-dimensional problems in infrastructure monitoring. The results show remarkable efficiency on currently tracked failure modes, extend current monitoring coverage and provide ways to interpret the results. These aspects are of paramount importance in a system which will need to be operated for years by field experts.

While the discussion was limited to specific datasets related to the CMS experiment, the applications are of general interest for high-energy physics experiments. Some of the proposed methods have already been integrated and deployed in the CMS DQM infrastructure. A generalization of these strategies could pave the way to full automation of the quality assessment for HEP experiments and accelerator complexes.

References

- [1] A. A. Pol, G. Cerminara, C. Germain, M. Pierini and A. Seth, Detector monitoring with artificial neural networks at the CMS experiment at the CERN large hadron collider, *Comput. Softw. Big Sci.* **3**(1) (2019) 3.
- [2] A. A. Pol, V. Azzolini, G. Cerminara, F. De Guio, G. Franzoni, M. Pierini F. Sirokỳ and J.-R. Vlimant, Anomaly detection using deep autoencoders for the assessment of the quality of the data acquired by the CMS experiment, in *EPJ Web of Conferences* (EDP Sciences, 2019).
- [3] D. Abbott *et al.*, Atlas data quality operations and performance for 2015–2018 data-taking, *J. Instrum.* **15** (2020) 04.

- [4] M. Adinolfi, D. Derkach, F. Archilli, A. Baranov, A. Panin, A. Pearce, A. Ustyuzhanin and W. Baldini, LHCb data quality monitoring, *J. Phys. Conf. Ser.* (2017).
- [5] B. von Haller *et al.*, The alice data quality monitoring, *J. Phys. Conf. Ser.* (2010).
- [6] M. Schneider, The data quality monitoring software for the CMS experiment at the LHC: Past, present and future, in *Proc. CHEP 2018* (2018).
- [7] CMS, V. Khachatryan *et al.*, The CMS trigger system, *JINST* **12**(01) (2017) P01020; arXiv:1609.02366 [physics.ins-det].
- [8] A. A. Pol, Machine learning anomaly detection applications to compact muon solenoid data quality monitoring, Ph.D. thesis, Université Paris-Saclay (2020).
- [9] C. Roderick, G. Kruk and L. Burdzanowski, The cern accelerator logging service-10 years in operation: a look at the past, present and future, Technical Report, CERN (2013).
- [10] L. Bottura, Cable stability, preprint (2014); arXiv:1412.5373.
- [11] R. Denz, Electronic systems for the protection of superconducting elements in the LHC, *IEEE Trans. App. Superconductivity* **16**(2) (2006) 1725.
- [12] J. Steckert and A. Skoczen, Design of fpga-based radiation tolerant quench detectors for LHC, *J. Instrum.* **12**(04) (2017) T04005.
- [13] D. M. Blei, A. Kucukelbir and J. D. McAuliffe, Variational inference: A review for statisticians, *J. Amer. Statist. Assoc.* **112**(518) (2017) 859.
- [14] C. C. Aggarwal, *Outlier Analysis*, 2nd edn. (Springer, 2016).
- [15] R. Chalapathy and S. Chawla, Deep learning for anomaly detection: A survey, preprint (2019); arXiv:1901.03407.
- [16] M. Goldstein and S. Uchida, A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data, *PLoS one* **11**(4) (2016) e0152173.
- [17] A. Zimek, E. Schubert and H.-P. Kriegel, A survey on unsupervised outlier detection in high-dimensional numerical data, *Statist. Anal. Data Mining: ASA Data Sci. J.* **5**(5) (2012) 363.
- [18] C. C. Aggarwal, A. Hinneburg and D. A. Keim, On the surprising behavior of distance metrics in high dimensional spaces, in *Proc. 8th Int. Conf. Database Theory, ICDT '01* (Springer, Berlin, 2001).
- [19] A. Hinneburg, C. C. Aggarwal and D. A. Keim, What is the nearest neighbor in high dimensional spaces? in *26th Int. Conf. Very Large Databases* (2000).
- [20] Y. LeCun *et al.*, Convolutional networks for images, speech, and time series, *Handbook Brain Theory Neural Netw.* **3361**(10) (1995) 1995.
- [21] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola and R. C. Williamson, Estimating the support of a high-dimensional distribution, *Neural Comput.* **13**(7) (2001) 1443.
- [22] F. T. Liu, K. M. Ting and Z.-H. Zhou, Isolation forest, in *Eighth IEEE Int. Conf. Data Mining, 2008. ICDM'08* (IEEE, 2008).
- [23] F. T. Liu, K. M. Ting and Z.-H. Zhou, Isolation-based anomaly detection, *ACM Trans. Knowledge Discovery Data* **6**(1) (2012) 3.

- [24] R. Shwartz-Ziv and N. Tishby, Opening the black box of deep neural networks via information, preprint (2017); 1703.00810 (2017).
- [25] D. Hendrycks and K. Gimpel, A baseline for detecting misclassified and out-of-distribution examples in neural networks, preprint (2016); arXiv:1610.02136.
- [26] G. E. Hinton, Connectionist learning procedures, in *Machine learning*, (Elsevier, 1990), pp. 555–610.
- [27] Y. Bengio, A. Courville and P. Vincent, Representation learning: A review and new perspectives, *IEEE Trans. Pattern Anal. Mach. Intell.* **35** (2013) 1798.
- [28] G. Alain and Y. Bengio, What regularized auto-encoders learn from the data-generating distribution, *J. Machine Learning Res.* **15**(1) (2014) 3563.
- [29] D. P. Kingma and M. Welling, Auto-encoding variational bayes, preprint (2013); arXiv:1312.6114.
- [30] D. J. Rezende, Stochastic backpropagation and approximate inference in deep generative models, in *Proc. 31st Int. Conf. on Machine Learning, ICML*, Vol. 32 (2014).
- [31] J. An and S. Cho, Variational Autoencoder based anomaly detection using reconstruction probability, Technical Report, SNU Data Mining Center (2015).
- [32] X. Wang, Y. Du, S. Lin, P. Cui and Y. Yang, Self-adversarial variational autoencoder with gaussian anomaly prior distribution for anomaly detection, preprint (2019); arXiv:1903.00904.
- [33] S. Zhao, J. Song and S. Ermon, Infovae: Information maximizing variational autoencoders, preprint (2017); 1706.02262
- [34] D. J. Rezende and F. Viola, Taming vaes, preprint (2018); arXiv:1810.00597.
- [35] E. Nalisnick, A. Matsukawa, Y. W. Teh and B. Lakshminarayanan, Detecting out-of-distribution inputs to deep generative models using a test for typicality, preprint (2019); arXiv:1906.02994.
- [36] J. Snoek, Y. Ovadia, E. Fertig, B. Lakshminarayanan, S. Nowozin, D. Sculley, J. Dillon, J. Ren and Z. Nado, Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift, in *Advances in Neural Information Processing Systems* (2019).
- [37] Y. Kawachi, Y. Koizumi and N. Harada, Complementary set variational autoencoder for supervised anomaly detection, in *IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)* (2018).
- [38] D. Hendrycks, M. Mazeika and T. G. Dietterich, Deep anomaly detection with outlier exposure, preprint (2018); arXiv:1812.04606.
- [39] K. Simonyan, A. Vedaldi and A. Zisserman, Deep inside convolutional networks: Visualising image classification models and saliency maps, preprint (2013); arXiv:1312.6034.
- [40] M. Borisyak, F. Ratnikov, D. Derkach and A. Ustyuzhanin, Towards automation of data quality system for cern CMS experiment, preprint (2017); arXiv:1709.08607.

- [41] A. Pol, V. Berger, G. Cerminara, C. Germain and M. Pierini, Trigger rate anomaly detection with conditional variational autoencoders at the CMS experiment, in *Machine Learning and the Physical Sciences Workshop at the 33rd Conf. Neural Information Processing Systems (NeurIPS)* (2019).
- [42] D. P. Kingma, S. Mohamed, D. J. Rezende and M. Welling, Semi-supervised learning with deep generative models, in *Advances in neural information processing systems* (2014).
- [43] K. Sohn, H. Lee and X. Yan, Learning structured output representation using deep conditional generative models, in *Advances in Neural Information Processing System* (2015).
- [44] M. F. Mathieu, J. J. Zhao, J. Zhao, A. Ramesh, P. Sprechmann and Y. LeCun, Disentangling factors of variation in deep representation using adversarial training, in *Advances in Neural Information Processing Systems 29*, eds. D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon and R. Garnett, (Curran Associates, Inc., 2016), pp. 5040–5048.
- [45] F. Locatello, S. Bauer, M. Lucic, G. Raetsch, S. Gelly, B. Schölkopf and O. Bachem, Challenging common assumptions in the unsupervised learning of disentangled representations, in *Proc. 36th Int. Conf. Machine Learning*, K. Chaudhuri and R. Salakhutdinov (PMLR, Long Beach, CA, 2019); <http://proceedings.mlr.press/v97/locatello19a.html>.
- [46] J. Lucas, G. Tucker, R. Grosse and M. Norouzi, Understanding posterior collapse in generative latent variable models.
- [47] M. Gemici, C.-C. Hung, A. Santoro, G. Wayne, S. Mohamed, D. J. Rezende, D. Amos and T. Lillicrap, Generative temporal models with memory, preprint (2017); arXiv:1702.04649.
- [48] S. A. Eslami *et al.*, Neural scene representation and rendering, *Science* **360**(6394) (2018) 1204.
- [49] K. Kawakami, Supervised sequence labelling with recurrent neural networks, Ph.D. dissertation (2008).
- [50] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.* **9**(8) (1997) 1735.
- [51] M. Wielgosz, A. Skoczeń and M. Mertik, Using LSTM recurrent neural networks for monitoring the LHC superconducting magnets, *Nucl. Instrum. Methods Phys. Res. Sec. A* **867** (2017) 40.

This page intentionally left blank

Part III

Generative Models

This page intentionally left blank

Chapter 6

Generative Models for Fast Simulation

Michela Paganini^{*,§}, Luke de Oliveira^{*,¶}, Benjamin Nachman^{*,||},
Denis Derkach^{†,**}, Fedor Ratnikov^{†,††},
Andrey Ustyuzhanin^{†,‡‡} and Aishik Ghosh^{†,§§}

^{}Lawrence Berkeley National Laboratory
1 Cyclotron Rd, Berkeley, CA 94720, USA*

[†]HSE University 20 Myasnitckaya Ulitsa, Moscow, Russia

*[‡]Université Paris-Saclay, CNRS/IN2P3,
IJCLab, 91405 Orsay, France*

[§][mpaganini](mailto:mpaganini@lbl.gov)

[¶][lukeoliveira](mailto:lukeoliveira@lbl.gov)

^{||}bpnachman@lbl.gov

*^{**}[dderkach](mailto:dderkach@hse.ru)*

^{††}[fratnikov](mailto:fratnikov@hse.ru)

^{‡‡}austyuzyhanin@hse.ru

^{§§}aishik.ghosh@cern.ch

Generative models are a class of machine learning methods that map random numbers into structured data. One potentially powerful application of generative models to high energy physics is as surrogate models for slow detector simulations. Interactions of particles with detector material is often the slowest component of the full simulation stack and so mimicking this component with a neural network may significantly accelerate the entire simulation. This chapter introduces various approaches to deep generative modeling and then provides examples of applications to high energy physics detector simulations.

1. Generative Models for the Simulation of Particle Showering in Calorimeters

Computational modeling of scientific phenomena, in particle physics as in other domains, represents a cornerstone of the knowledge-building process of hypothesis formulation and testing. Through

model building and probing, scientists can generate expected outcomes and compare them with experimental results. Theoretical models of natural phenomena, in order to stand the test of time, ought to make testable predictions that match the observed manifestations of the laws they propose.

Of peculiar interest to particle physicists is the continued probing of the Standard Model (SM) of Particle Physics for possible discrepancies and inconsistencies with observations, and the testing of Beyond the Standard Model (BSM) theories that expand this model to include new symmetries, interactions, and particles. Event generators describe and simulate new and known fundamental physics interactions, and allow to produce and study virtual particle collisions and decays.

In real laboratory settings, particle detectors are constructed to measure particle properties through means of controlled interactions with the detectors' material. In the context of this book, this section will focus on the specific use-case of modeling and simulating the low- and high-energy interactions of particles with matter — a phenomenon that underpins the processes of detection and identification of these particles with detectors.

A detailed geometric and physical description of particle detectors is traditionally used to simulate the response of these instruments to their interaction with particles that propagate within their volumes to guide their design process, optimize their layout and readout properties, and quantify their expected effect on downstream data analysis tasks. The physics processes of interest to be targeted with the construction or upgrade of an existing detector dictate the objectives and constraints on the properties of the apparatus. To study the suitability of a hardware solution towards the measurement of a physics signal, as well as the ability to extract that signal from the noise of background processes with similar signatures, physicists resort to simulating these events within virtual representations of the instrument. Furthermore, simulation of particle interactions within existing detectors, and their comparisons with the observed readouts captured by the corresponding, real hardware detectors, can be used

to measure and account for effects of physical degradation in the instrument and improve the likeness of simulators.

Given the large dynamical range of particles of interest that traverse the detector volumes at particle experiment sites, and given the prodigious physical scale of these machines, high-fidelity simulation of all interactions, which are testimony of the occurrence of physical processes governed by fundamental physical laws of nature, requires immense computational power for the stochastic description of these events, with up to minutes spent on each event in simulations at the large hadron collider (LHC) [1]. Among the main drivers for optimization and modernization of the simulation software is the increased luminosity expected in the next runs and phases of the LHC, during which the demand for the number of simulated events necessary to support the goals of the physics program is expected to grow to beyond the current reach of the full simulation infrastructure.

Surveys of the computational costs of the various simulation stages in the pipeline of typical LHC experiments have identified the accurate generation of particle showers in calorimeters as one of the most compute-intensive phases, and, therefore, as an ideal candidate for the development of fast simulation techniques to approximate it [2]. In the ATLAS experiment, for example, calorimeter full simulation accounts for approximately 90% of total simulation time [3]. Simulated sample storage concerns have also been raised and presented as arguments for the development of novel fast simulation techniques that can be evaluated on the fly.

Traditional multi-purpose simulation packages, such as the GEANT4 toolkit [4], are used to accurately model, to a meticulous extent, detector responses and energy depositions in calorimeters, produced by the interactions of particles with the material in the detectors. Despite themselves being subject to continued improvement, these methods, often referred to as *full simulation* approaches, are considered as the ground truth for the interactions they model, as they encode current physics knowledge with maximal fidelity. In opposition, *fast simulation* techniques (see Sec. 1.2) attempt to approximate the data distribution generated by full simulators with

various degrees of fidelity, depending on the nature of the speed vs. precision trade-offs they make.

The following will serve as an introduction to generative modeling with machine learning, and provide concrete examples of how deep generative modeling can be employed towards the goal of speeding up physics simulation. Section 1.1 provides an overview of current techniques in deep generative modeling. Then, Sec. 1.2 introduces fast calorimeter simulations. Deep generative models for such simulations are introduced for one-layer detectors in Sec. 1.3 and multi-layer detectors in Sec. 1.4. A brief discussion of biases in generative models is presented in Sec. 1.5 and the chapter ends in Sec. 2.

1.1. *Generative models*

Generative modeling aims to learn and emulate the process by which data are generated according to some true, unknown generating distribution $p_{\text{data}}(x)$ for features $x \in \mathcal{X}$, or $p_{\text{data}}(x, y)$, if labels y are known, i.e., to approximate the distribution $p_{\text{model}}(\theta) \approx p_{\text{data}}$.

Unlike discriminative models, that only learn the conditional distribution $p(y|x)$ through a direct input-to-label mapping, generative models are assigned the more complex task of modeling the full data-generating distribution. If the labels are known, and once an approximation to $p_{\text{data}}(x, y)$ has been obtained, Bayes theorem allows to repurpose the generative model and transform it into a classifier. Generative models can also be used to ascertain distributional parameters of interest and corresponding confidence intervals, in the case in which they admit a closed form likelihood.

In practice, the true data distribution is usually opaque: it is not available in analytical form, but only as an empirical distribution, through a finite amount of observations distributed according to it. Sampling new examples from the true data distribution is either inaccessible or expensive. The objective of generative modeling, then, is to approximate said distribution, to enable infinite sampling from it for practical applications.

Depending on whether they define an explicit (tractable or approximate) or implicit density, generative modeling techniques can

be subdivided into categories of methods that share commonalities in their approach.

A first framework to analyze the learning procedure of a generative model is maximum likelihood estimation (MLE), which aims, as the name suggests, to maximize the likelihood of the observed data under a model assumption. Formally, when it is possible to construct a parametric model $p_{\text{model}}(x; \theta)$, parameterized by θ , MLE aims to maximize $\mathcal{L}(X; \theta)$ defined as follows:

$$\mathcal{L}(X = \{x_i\}_{i=1}^n; \theta) = \prod_{i=1}^n p_{\text{model}}(x_i; \theta). \quad (1)$$

In practice, direct optimization of the likelihood can happen, for example, via MCMC, and, for numerical stability purposes, it is preferable to minimize the negative log likelihood: $\theta^* = \arg \min_{\theta} \{-\ln \mathcal{L}(X; \theta)\} = \arg \min_{\theta} \{-\sum_{i=1}^n \ln p_{\text{model}}(x_i; \theta)\}$, as opposed to maximizing a product of likelihoods.

Hand-crafted parametric models, although tractable and interpretable, often lack in capacity and expressivity. On the other hand, deep generative models offer increased flexibility through a wide and diverse range of approaches, but may do so by removing the advantage of a tractable density.

Auto-regressive models (such as PixelRNN and PixelCNN [5], WaveNet [6], GPT [7], and XLNet [8]) are a powerful family of generative models that still admits a tractable joint likelihood, which is explicitly factorized into the product of successive conditional likelihoods of each input feature: $p(x) = p(x_1) \prod_{i=2}^d p(x_i | x_{i-1}, \dots, x_1)$, for $x \in \mathbb{R}^d$. Concretely, this corresponds to generating the example one feature at a time, conditioned on previously produced features. To model complex, long-range correlations, expressive RNN-based and CNN-based sequence models are often employed for this task. They have shown remarkable performance on benchmark applications in industry, especially on discrete output spaces, where competing techniques fail. The fundamentally sequential nature of their generation process, however, makes them hard to parallelize for efficient generation in inference mode. Other generative modeling techniques that admit tractable formulations and make use of the

auto-regressive approach include neural auto-regressive distribution estimation (NADE) [9] and masked auto-encoders for distribution estimation (MADE) [10].

In contrast with *fully visible models*, the family of generative models that assumes the presence of unobserved variables z governing the distribution of occurrences takes the name of *latent variable models*. The goal of a latent variable model is to learn a map between the lower-dimensional representation in latent space and the high-dimensional representation in observable space. Modelers hypothesize that the complexity of observed high-dimensional data can be reduced into low-dimensional, sufficient embeddings. This compression, or encoding, step yields more fundamental, efficient representations of the data that live in a lower-dimensional manifold. Traversing this manifold allows to explore the full range of diversity in the data. Enforcing semantically meaningful latent representations can enable the traversal of the latent manifold along intelligible and interpretable directions, thus providing explicit handles to control the generation process. Obtaining disentangled factors of variation is the focus of much of the ongoing research in the machine learning literature.

Invertible flow models [11, 12] also take an explicit density estimation approach, but make use of normalizing flows [13, 14] to render the problem tractable, while retaining the ability to model complex distributions. In practice, through a series of invertible transformations, flow-based models allow to move from input to output representation in a bidirectional way, by directly modeling $\hat{x} = f^{-1}(f(x))$, where f is compositional in nature. Their main practical disadvantage is said to reside in the inefficient computational cost at training time, with additional concerns around the suitability and expressivity of normalizing flows in approximating desirable probability functions [15]. Many models are interrelated — the initial features x in invertible flow models can be viewed as a useful (albeit not lower-dimensional) latent space.

In fact, many commonly used deep generative models (e.g. variational auto-encoders (VAEs) [16] and generative adversarial networks (GANs) [17]) are not directly optimized via MLE, but do

have interpretations which cast them as consistent with MLE. When the likelihood is not accessible, distribution comparison metrics d can be used to reformulate the learning problem as $\theta^* = \arg \min_{\theta} \{d(p_{\text{data}}(x, y), p_{\text{model}}(x, y; \theta))\}$.

Popular formulations have phrased the generative optimization problem in terms of maximum mean discrepancy [18–20], or score matching [21].

In the field of artificial intelligence (AI) research, recent innovations in generative modeling have brought this topic to the forefront of the academic discourse, both from the theoretical and the empirical perspective. Qualitative achievements of generative models, brought upon by new algorithmic developments, have been accompanied by a wealth of theoretically grounded propositions, geared towards explaining the properties and improving the stability, efficiency, performance, and scale of these models. Among the AI applications of generative modeling, we find, for example, image in-painting, anomaly detection, missing data imputation, data augmentation, image-to-image translation, domain adaptation, latent representation learning, planning, up-sampling, and super-resolution.

Though a criticism of the current research trends around generative modeling in the AI field is that authors may have lost sight of the goals of generative modeling, physics and other scientific domains may yet come to the rescue, by providing a fertile soil for the application of these models. In fact, generative models’ achievements have allowed to them to make the jump to applied and fundamental scientific domains, with successful application to science problems from high-energy physics (GANs [22–58], AEs [37, 59–61], physics-inspired [62–64], normalizing flows [65–73], mixture models [74], phase space [68–70, 75–80]), climate science, cosmology, material science, and more [81–84]. In these application domains, generative models hold promise as tools for emulating, with high fidelity, the performance of computationally intensive scientific simulators. Their utility has primarily been that of providing a lifeline for many statistically and computationally bound problems that would benefit from increased data availability.

1.1.1. Generative adversarial networks

Generative adversarial networks (GANs) [17] are implicit density models that do away with directly modeling or approximating a probability density function, and instead offer ways to indirectly interact with it, for example, through sampling. While this may limit their applicability, it renders them particularly well suited for problems requiring efficient querying of the data distribution for new examples.

As shown in Fig. 1, the idea behind GANs is to pair a generator, that transforms a latent code into an observable representation, with a competing network, that automatically scrutinizes the quality of the generator-fabricated samples by learning to distinguish them from real samples from the target distribution. The presence of the discriminator is merely a training artifice, with the scope of automating quality assurance. Intuitively, the generator's objective is to produce realistic-looking samples that even a well-trained discriminator may confuse for real. Ideally, in a figurative sense, the two players would engage in an adaptive process of iterative refinement of the generated samples, with the discriminator adapting to new features and patterns explored by the generator, and providing signal to the generator for further improvement.

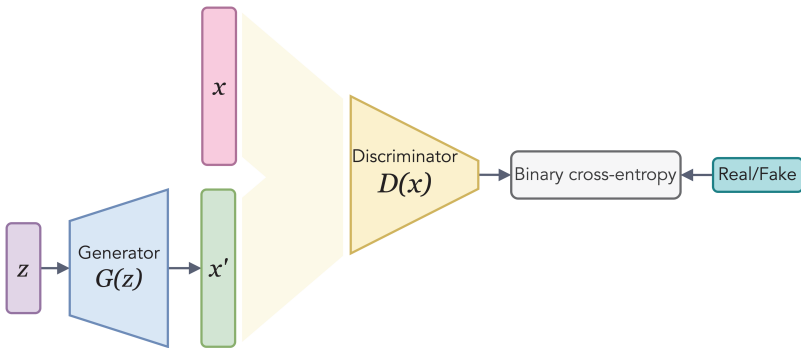


Fig. 1. Schematic of a generative adversarial network (GAN). A latent space random variable Z is sampled to produce examples z that are then mapped through a generator function $G(z)$ to produce examples x' that are supposed to statistically match examples x drawn from the random variable X . A classifier D is trained to distinguish sets of x' and x to provide feedback to the generator.

Formally, GANs define a setting in which a pair of deep neural networks participate in a non-cooperative minimax game in order to implicitly learn to approximate a true, unknown data distribution $X \sim p_{\text{data}}$ that generates training examples x . The *generator* network maps a latent prior $Z \sim p_Z$, $Z \in \mathcal{Z}$ to the observable space (such as the pixel space for image generation) of the distribution, \mathcal{X} ; usually $Z \sim \mathcal{N}(0, \mathbb{I})$ or $Z \sim \mathcal{U}([0, 1]^n)$ (uniform distribution on a hypercube). A second network, the *discriminator*, attempts to learn a mapping from \mathcal{X} to a probability of a given sample $x \in \mathcal{X}$ being real, i.e. having originated from p_{data} and not having been fabricated by the generator. The generator G parameterized by parameters θ_G provides a map $G(\cdot; \theta_G) : \mathcal{Z} \rightarrow \mathcal{X}$, while a discriminator D parameterized by parameters θ_D provide a map $D(\cdot; \theta_D) : \mathcal{X} \rightarrow [0, 1]$.

p_{data} is implicitly learned via a procedure in which G and D are trained concomitantly in a zero-sum minimax game with the total objective defined as:

$$\min_{\theta_G} \max_{\theta_D} \underbrace{\mathbb{E}_{X \sim p_{\text{data}}} [\log D(X; \theta_D)]}_{\text{log probability of the discriminator perceiving real samples as real}} + \underbrace{\mathbb{E}_{Z \sim p_Z} [\log(1 - D(G(Z; \theta_G); \theta_D))]}_{\text{log probability of the discriminator perceiving generated samples as fake}} \quad (2)$$

From a distribution comparison standpoint, the choice of using a log value function in Eq. (2) corresponds to minimizing the Jensen–Shannon divergence between p_{model} and p_{data} [17].

In practice, GAN training may proceed by simultaneous weight update or by successive, alternating gradient descent and ascent steps (similar to block coordinate descent) to iteratively update the discriminator and generator networks. The amount of emphasis placed on each step and, consequently, the ratio of training iterations spent on each network, have both practical and theoretical implications for convergence [85, 86]. Nonetheless, convergence guarantees in the unmodified GAN game in Eq. (2) are only local and valid under mild to strict assumptions of solution existence and proximity [87]. By studying the game’s vector field, one sees that the dynamics of GAN training often exhibit orbiting patterns around locally stable stationary points along the way to a saddle point in the loss function [88–90].

In general, in fact, the formulation defined in Eq. (2) exhibits well-documented issues with respect to stability, saturation, and divergence [17, 89], leading to practical and theoretical improvements [91] involving neural network architecture design [92, 93], auxiliary tasks [94], improved objective functions and distributional divergence measures such as Wasserstein distances [95–100], and modifications to the gradient descent algorithm [88, 101]. Recent salient improvements, most of which are highly empirical in nature, achieve near-photo-realistic quality when applied to the image domain (Fig. 2), by leveraging practical enhancements, such as staged generation at increasing resolution [93] and larger scale training and architectures [102].

In practice, training GANs requires a number of tricks and empirical modifications to induce stability and better-posed learning dynamics. To avoid gradient saturation and improve training dynamics, label flipping and label smoothing, especially when applied in early iterations of training, help ensure that the discriminator does not converge too early causing no gradient to carry to the generator [97]. Label flipping, with some small probability, flips the 0 and 1



Fig. 2. 512×512 generated samples from BigGAN [102].

(fake and real) labels, while label smoothing replaces positive labels 1 with a smoothed value α , usually either sampled or set to a fixed value like 0.9. Adding instance noise represents yet another regularization technique to empirically stabilize GAN training by augmenting the support of the true and generated distributions by convolving them with a tunable noise distribution [103].

A commonly documented failure mode in GANs is so-called *mode collapse*, where the generator learns to produce samples with little variety from a small, finite region in the support, thus only modeling a limited number of modes in the true multimodal distribution. Intuitively, this is a form of short-term exploitation of each player's weaknesses, by which the generator resorts to producing limited kinds of samples that maximally fool only the current version of the discriminator, and the discriminator learns to be optimal only with respect to the current version of the generator, resulting in an undesired loop. To discourage this behavior, one way of addressing the problem is to allow the discriminator to make use of batch-level statistics to build a more robust understanding of the global probability distribution and sample diversity, in a process called minibatch discrimination [97]. Other fixes suggest, for example, providing the networks with a replay buffer to prevent the forgetting of previously explored modes [104], introducing spectral regularization to prevent the correlated spectral collapse [105], or adding a further autoencoding network and loss term, as suggested in VEEGAN [106].

In degenerate cases (which are common in early training iterations) where the implicit distribution from the model and the data distribution are disjoint, even the original non-saturating heuristic modification of the value function [17], which addresses the problem of lack of gradient feedback for highly improbably fabricated samples, still suffers from vanishing and exploding gradients. A more principled approach is provided by Wasserstein GAN [95] (WGANs), which defines a distributional distance measure that is well suited to cases of disjoint support between the generator and the data distribution. WGAN recasts the discriminator D as a *critic*, and designs the inner loop of the critic training to learn to approximate the Wasserstein-1

distance, or Earth-Movers distance, between p_{data} and p_{model} :

$$W(p_{\text{data}}, p_{\text{model}}) = \inf_{\xi \in \Pi(p_{\text{data}}, p_{\text{model}})} \mathbb{E}_{X, \hat{X} \sim \xi} \|X - \hat{X}\| \quad (3)$$

$$\begin{aligned} &= \sup_{\|D(\cdot; \theta_D)\|_L \leq 1} \mathbb{E}_{X \sim p_{\text{data}}} [D(X; \theta_D)] \\ &\quad - \mathbb{E}_{Z \sim p_Z} [D(G(Z; \theta_G); \theta_D)]. \end{aligned} \quad (4)$$

The critic update step in WGAN amounts to simply maximizing the right-hand side of Eq. (4), while the generator seeks to minimize this value after a critic update. The critic is the analog of the discriminator in the standard GAN described earlier, only that the critic does not classify examples as real or fake. Instead, the critic approximates a distance between densities. Note that the Kantorovich–Rubenstein duality [107] is used to go from the intractable computation in Eq. (3) to the tractable one in Eq. (4), and this introduces the constraint on the discriminator to have Lipschitz constant less than or equal to one. To enforce a bounded Lipschitz constant in a neural network, the original WGAN formulation suggests clamping the weights of the neural network to be within some compact range $[-c, c]$. Later work, which introduces the Gradient Penalty WGAN [96] (WGAN-GP), proposes the addition of a regularization term to the learning, encouraging the magnitude of the gradient of $D(\cdot; \theta_D)$ to be close to 1 in the neighborhood of samples. Spectral Normalization [108] takes this one step further, utilizing partial power iterations during training to normalize all weights in the network, and causing the entire network itself to have Lipschitz constant equal to 1.

Despite their many practical difficulties (mode collapse and more) and subsequent re-formulations (WGAN and other variations), the popularity of GANs stems primarily from their unique game-theoretical formulation, their pragmatism and readiness for deployment in data augmentation systems, the high quality and superior realism of their samples, and their inexpensive inference mode computation. Empirically, their demonstrated competitiveness in fitting complex, high-dimensional, continuous distributions have made them a preferred tool in many application domains. Their primary drawbacks include instability, mode-collapse, and other optimization and

trainability issues, in addition to the difficulty of quantitatively evaluating the performance of these models in unequivocal ways.

As discussed, empirically observed challenges include GANs' inability to reach solutions that approximate distributions with full support [109]. However, thanks to the parameterized nature of the true data distributions often available in scientific applications, along with semantically meaningful marginal distributions, physics may represent a promising domain for algorithmic improvement in generative modeling, aimed at addressing this flaw. The exceptional control over training dataset synthesis that the sciences enjoy positions these fields as likely catalysts for innovation and creative algorithmic thinking, beyond what is possible with traditional, natural AI datasets collected in the wild.

Adversarial training has further been considered, within the scope of generative models, as a powerful ingredient in hybrid models, such as Boltzmann encoded adversarial machines [110] and adversarial auto-encoders [111].

1.1.2. Variational auto-encoders

Auto-encoders are explicitly tasked with encoding the high-dimensional observable data $x \in \mathcal{X}$ into a convenient low-dimensional latent representation, or code, $z \in \mathcal{Z}$, and then, in a second stage, decoding it back into the high-dimensional space to faithfully reconstruct the input as $\hat{x} \in \mathcal{X}$. Intuitively, the transition from high to low dimensionality, and back, forces the network to efficiently compress the essential information through the information bottleneck, while simultaneously learning the precise mapping that transforms latent codes into the data representations from the dataset. The dimensionality of the latent space plays a crucial role in turning what could be a trivial task of identity mapping into a potentially impossible task, when the bottleneck is too narrow.

Auto-encoder architectures, as seen in Fig. 3(a), can be split into two halves: an encoder $f : \mathcal{X} \rightarrow \mathcal{Z}$, that maps the input to the latent code, and a decoder $g : \mathcal{Z} \rightarrow \mathcal{X}$, that maps the latent code to a reconstructed version of the input, \hat{x} .

Traditionally, auto-encoders are trained via back-propagation with a squared reconstruction loss $\mathcal{L}(x, x') = (x - x')^2$ between the input x and the reconstruction output x' .

Variational auto-encoders (VAEs) [16], which share the underlying architecture of auto-encoders, take an approximate, variational approach to the intractable task of generation via maximum likelihood. These probabilistic models approximate posterior distributions in a differentiable way through neural networks, by maximizing the variational lower bound, or evidence lower bound (ELBO), of the likelihood [112]. In other words, instead of imposing a bottleneck with a restricted latent space dimension/encoder expressivity, a VAE creates an information bottleneck by imposing a particular form for the latent space probability density.

Unlike classical auto-encoders, VAEs have the ability to sample from the latent space, prior to decoding, as is shown in Fig. 3. In the encoder phase, the encoder network, parameterized by weights θ_E , learns an approximate posterior distribution $q_{\theta_E}(z|x)$, over which we have a prior $p(z)$. In the decoder phase, the decoder network, parameterized by weights θ_D , learns a likelihood $p_{\theta_D}(x|z)$. To enable sampling in latent space, it is convenient to impose a particular functional form (and to be able to compute the KL divergence analytically) to the prior $p(z)$ over the latent variables. This can oftentimes be a

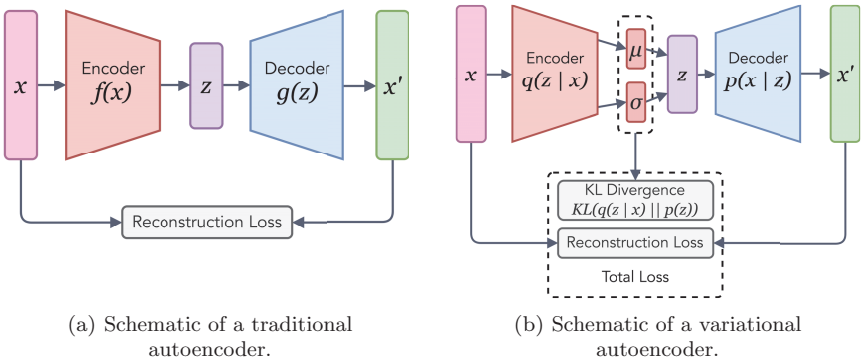


Fig. 3. Pictorial representations of traditional (left) and variational (right) auto-encoder architectures.

Gaussian prior, for example. The loss used to train the system,

$$\mathbb{E}_{Z \sim q_{\theta_E}, X \sim p_{\text{data}}} [\mathcal{L}(X, g_{\theta_D}(Z)) + \text{KL}(q_{\theta_E}(Z|X) \parallel p(Z))], \quad (5)$$

is then comprised of two terms: a reconstruction loss and divergence loss (Fig. 3(b)). The former ensure proximity between the input x and the reconstructed output $\hat{x} = g_{\theta_D}(z)$. The latter provides a regularization constraint on the learned, approximate posterior distribution of latents, to ensure proximity with a distribution of desirable functional form from which it is possible to sample efficiently.

To allow for back-propagation through non-differentiable expectations introduced by the stochastic sampling of the latent code, the code z can be further reparameterized in terms of auxiliary random noise $\epsilon \sim \mathcal{N}(0, \mathbb{I})$, for example, for a Gaussian prior, such that $Z = \mu + \sigma \cdot \epsilon$ for $Z \sim \mathcal{N}(\mu, \sigma^2)$.

1.2. *Fast simulation of particle showering in calorimeters*

To counter the growing compute requirements to run state-of-the-art detector simulation with GEANT4 [4], fast simulation techniques have been devised to produce adequate, simplified approximations of the data distribution produced by portions of the simulation pipeline. These methods target applications and studies that require large simulated datasets but do not demand a detailed description of the detector volume and response. Viable use-cases for fast simulation include, for example, large BSM parameter scans and certain detector upgrade studies [2].

Different speed-accuracy trade-off levels are required in different scenarios, leaving room for multiple fast simulation solutions that target specific parts of that spectrum. The Delphes package, for example, popular for phenomenological studies, applies simple smearings to generator-level particle four-vectors, and affords several orders of magnitude speed-ups [113].

In experiments like ATLAS and CMS, common fast simulation techniques for particle showers in the calorimeters make use of pre-computed frozen showers, parameterized showers, and lookup tables

for low-energy interactions [114, 115]. Several approximations are usually introduced in fast calorimeter simulation, including geometric simplifications, analytical descriptions of material interactions, and separate parameterizations of the longitudinal and transverse energy depositions and shower shapes [116–118]. In ATLAS, for example, parameterizations are derived from electron and photon showers, in the electromagnetic case, and charged pion showers, for the hadronic case, that are fully simulated with GEANT4. For improved representation, ATLAS’s FastCaloSimV2 utilizes principal component analysis to model the correlations of the energy depositions in each calorimeter layer [119]. Further corrections are applied to account for the artifacts introduced by the coarsification of the detector structure into cuboids [119]. While electromagnetic shower modeling reaches adequate levels of fidelity in shower shape description, the more complex simulation of hadronic showers has pushed researchers to develop techniques to model and correct for the discrepancy, using, for example, among others, VAE-based approaches [116]. In the specific case of the ATLAS experiment, a dedicated program for fast simulation of the tracker (FATRAS [120]) has been developed in parallel to the calorimeter-focused effort (FastCaloSim [121]). The CMS experiment has created its own fast simulation package, known as CMS FastSim [118]. The inclusion of these fast simulation routines has been shown to decrease the total simulation time by 1 or 2 orders of magnitude for a variety of physics processes, compared to full simulation [3, 117]. An advantage of customized fast simulation software is its seamless integration with other portions of an experiment’s simulation pipeline, such as digitization and reconstruction, along with the compatibility of formats and interfaces [122]. Fast simulation of specific detector portions are usually accompanied by full simulation of other detector components in hybrid simulation workflows.

Though faster than detailed Monte Carlo simulation, such approaches may suffer from a lack of realism and detail, especially in certain portions of the detector or of the particle’s phase space. In addition, fast simulation still encounters difficulties in modeling the sub-structure of energy depositions. This directly impacts the

downstream ability to apply findings from simulation to data, or to use fast simulated datasets to build reliable expectations of observed phenomena, especially for tasks and analyses that require a high fidelity description of, e.g. jet sub-structure [122].

If the simulation of particle interactions with detectors is recast as a sampling problem, where $p_{\text{data}}(x)$ is opaque and very expensive to sample from, generative models, as discussed in Sec. 1.1, lend themselves towards being able to act as a surrogate in order to draw new samples in a quick and efficient manner. Traditional fast simulation methods can be viewed through this lens, with lookup tables functionally acting as a non-parameteric generative model.

1.3. Deep generative modeling for fast simulation of jets in a 1-layer calorimeter

Although calorimeters often display a great degree of complexity in their hardware geometry, with multiple stacked layers of different depth, granularity, material, and orientation, it is useful to begin the exploration of deep learning-based generative modeling for calorimeter simulation from elementary, planar detector configurations.

Reference [123] opens the path towards fast simulation with deep generative models, by first showing the potential of GANs applied to the generation of particle energy deposits in a single layer LHC-like calorimeter. It introduces an architecture called location aware generative adversarial networks (LAGAN), capable of learning to generate idealized representations of jets known as jet images [124] (see Fig. 4), in a class-conditional manner, building on Auxiliary Classifier GANs [94]. Such a representation is a natural stepping stone towards more realistic calorimeters, given the discretized grid structure.

LAGAN makes use of locally-connected layers to learn location-dependent filters for dedicated processing of jet core and periphery, thus capturing key substructure information for the characterization of different particle jet types. Unlike popular GAN architectures in AI research, LAGAN chooses rectified linear activations to induce the desired level of sparsity in the output representation.

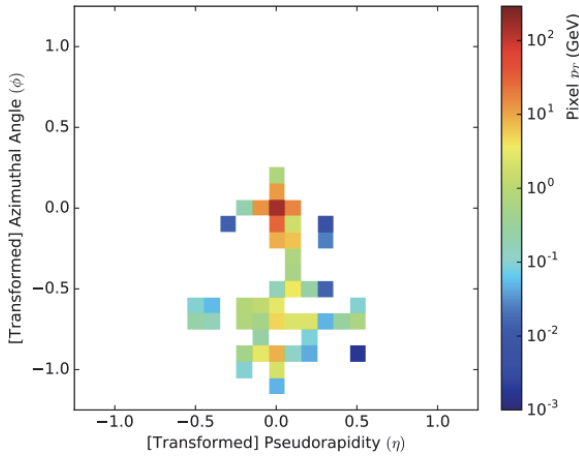


Fig. 4. An example of a typical jet image. Note the discretization of $\eta - \phi$ space with per-pixel energy deposits. Figure reproduced from [123].

Though modeled under idealized conditions, the simplified setting of this early work demonstrates the following:

- deep generative models, and GANs in particular, are capable of handling the high-dynamic range of energy depositions from particle showers (see Fig. 6(a));
- such models are capable of handling high levels of sparsity, even in idealized sensor readout (see Fig. 4);
- important physics quantities (reconstructed p_T , mass, etc.) are consistent with traditional simulation (see Fig. 6(b)).

Results from this early work show promise in reconstructing the substructure and kinematic features of hadronic jets. Figure 6(a) shows the dynamic range and distributional similarity between GAN-generated and Pythia-generated [125] training images with respect to pixel intensities, obtained from [123]. Figure 6(b) shows the class-conditional distributions of calculated jet mass between GAN-generated and Pythia-generated jet images, showing the ability of the LAGAN model to provide class-conditional sampling between signal ($W' \rightarrow WZ$) and background (QCD dijets). The code and datasets used in this work have been made publicly available [126, 127].

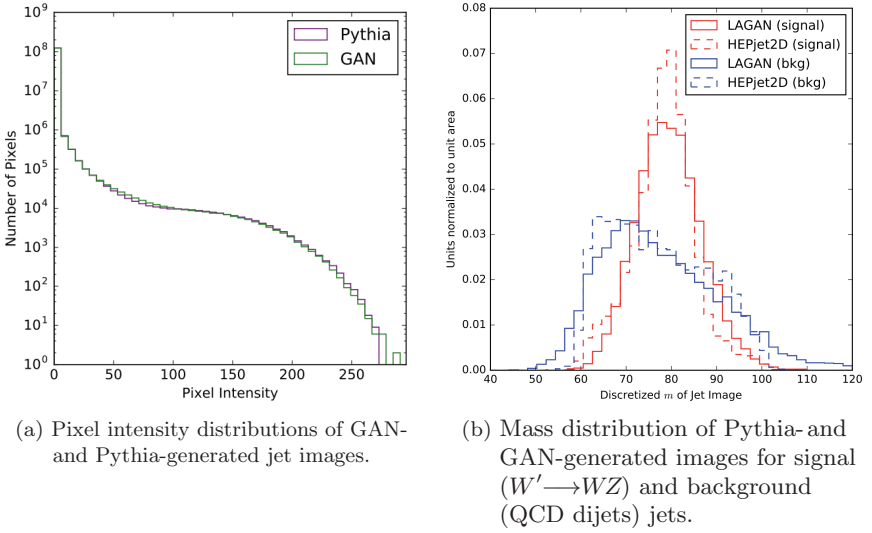


Fig. 5. GAN-generated cosmic proton event interacting with ground-based water-Cherenkov detectors. Figure reproduced from [128].

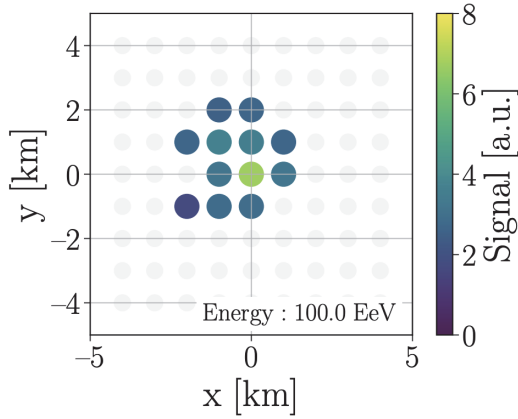


Fig. 6. Pixel intensity distribution and mass distributions of GAN- and Pythia-generated jet images. Figures reproduced from [123].

Reference [128] opts to use a WGAN and expands scope to show utility in both particle and astroparticle physics, by simulating a ground-based array of detectors to observe cosmic ray-induced air showers (see Fig. 5 for an exemplar of sampled sensor read-out

from this setup). Similar to [129], [128] explicitly encodes physics knowledge in the form of domain-specific constraints in the loss formulation.

1.4. *Deep generative modeling for fast simulation of electromagnetic showers in multi-layer calorimeters*

The peculiar geometry of many calorimetry detectors at particle colliders demands the construction of generative shower models that are able to output representations with commensurate dimensionality to that of the physical detector hardware. Detectors at the LHC, for example, possess multiple concentric types of multi-layer calorimeters with irregular, heterogeneous volume segmentation into cells of varying size. Physics considerations, towards the optimization of the measurement of particle shower properties, drive the design choices that go into the manufacturing of these apparatuses. Each calorimeter layer and configuration is designed to have different depth and granularity. While a single-layer calorimeter simulator may suffice for many of the detectors currently found at particle colliders, addressing the needs of experiments like requires moving beyond the single-layer assumption and the associated planar image representation, typical of LAGAN-like model architectures.

To increase realism beyond single-layer calorimetry detectors, solutions that examine the added complexity of generating realistic electromagnetic shower signatures in multi-layer calorimeters have proposed GAN- and VAE-based approaches capable of handling the dimensionality increase [130–132]. Many of these still introduce simplifications to the full detector geometry by focusing on central regions of detectors and removing edge cases that would require special treatment.

A natural tendency in such an application domain is to tailor models to data formats and corresponding detector representation. Therefore, the foundational literature in generative modeling for multi-layer electromagnetic shower simulation can be thought of as being primarily differentiated among data representation axes. Most creativity in model design, then, remains directly bound to the

explicit inductive bias considerations that drive data structure and representation selection.

Two main approaches to deep generative modeling for this problem emerge as a result of the piqued interest of the community in fast simulation using GANs. Inspired by different kinds of calorimeters found across particle detectors, GAN-based contributions either use heterogeneous detector geometry [130, 131], and thus require explicit per-layer modeling, or use uniform detector geometry, and are able to leverage volume-appropriate architectural components such as 3D convolutions [132, 133]. Additional work compares the fidelity of GAN-based vs. VAE-based learning to simulate calorimeter response to particle propagation in the ATLAS detector at CERN, and evaluates the fidelity of the data produced by these deep generative models to full GEANT4-based ATLAS simulation [37, 134]. For both computational and phenomenological reasons, all approaches model the development of a single particle shower at a time, and then exploit the compositionality property of showers to later assemble full events.

References [130, 131] demonstrate the utility of a GAN-based fast simulator on electrons, positrons, and charged pions, in an idealized, ATLAS-like multi-layer calorimeter. The design of CALOGAN explicitly encodes energy conservation and conditioning into the training, models the inherently heterogeneous geometry of the three calorimeter layers through three separate network components, and incorporates cross-layer energy correlations and conditioning through the usage of layer-to-layer attention. As is shown in Fig. 7, CALOGAN is able to produce highly realistic, yet unseen, calorimeter showers that add to the diversity of possible shower outcomes in the irregular geometry. The code and datasets used in this work have been made publicly available [135, 136].

References [132, 133] propose GANs for calorimetry simulation of homogeneous detector volumes, using neural architecture components such as 3D convolutions. This is best suited for the representation of regular geometries, and directly builds upon the LCD CLIC detector design [137], which results in an effective output volume of dimension $51 \times 51 \times 25$. Example transverse slices are shown in Fig. 8. In addition to displaying good agreement between GEANT4 and the

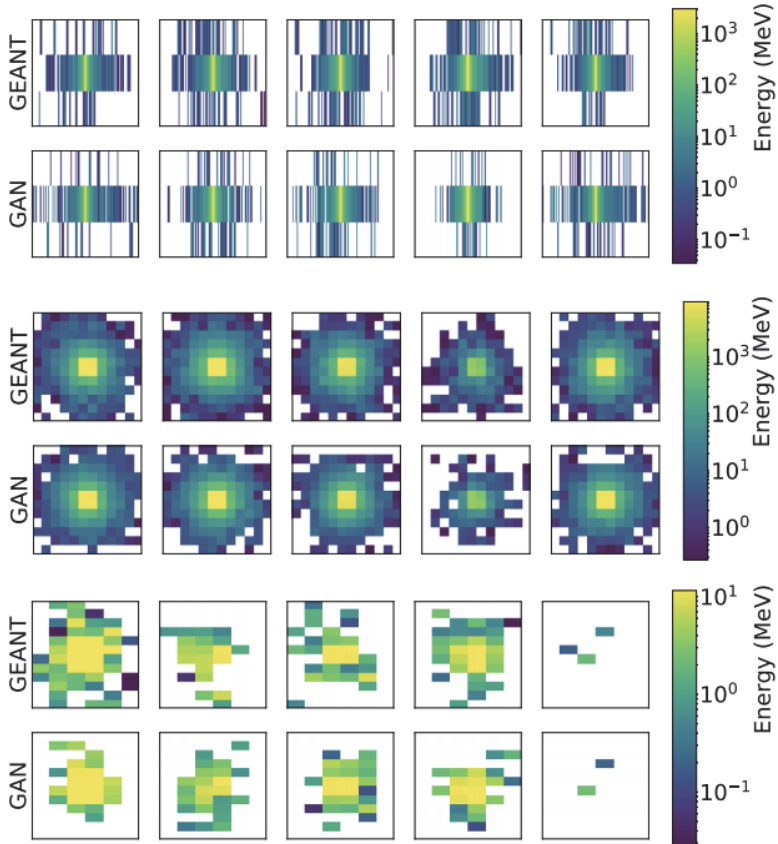


Fig. 7. Five randomly selected positron showers per calorimeter layer from the GEANT4 (top) and the five nearest neighbors generated from CALOGAN. Figure reproduced from [130].

3D-GAN learned simulation on a qualitative level, the authors rely on downstream physics observables like cross-sectional energy profiles and positions to evaluate the efficacy of their proposed approach. Thanks to the homogeneous detector structure, the 3D convolutions prove to be a powerful tool for modeling propagation through this medium.

In another effort to anticipate the role that efficient AI-driven simulation (as well as reconstruction) will play at future colliders, [54] explores the use of energy-conditioned bounded information

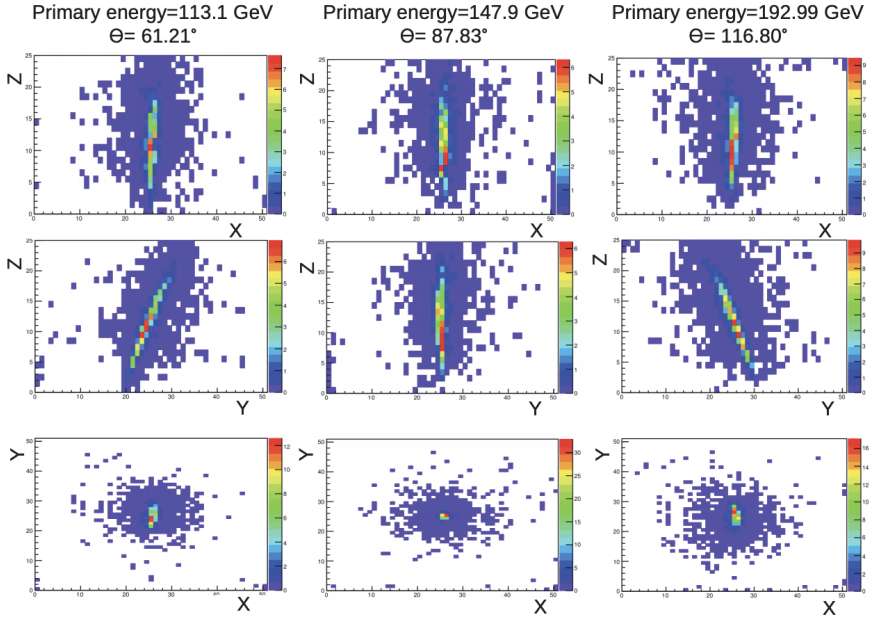


Fig. 8. Slices of GAN-generated electron showers. Figure reproduced from [133].

bottleneck autoencoders (BIB-AE) [138] to emulate the GEANT4 production of electromagnetic showers in the central region of the high-granularity Silicon-Tungsten calorimeter of the proposed International Large Detector [139]. In these experiments, through the addition of a maximum mean discrepancy (MMD) term to the loss, previously employed for LHC event generation [140], the BIB-AE outperforms other modeling choices considered, such as GANs and WGANs, with important improvements over the modeling of the energy deposition of minimum ionizing particles. This work further demonstrates the aptitude of deep generative models to recover physically meaningful differential distributions with high fidelity in yet another calorimeter geometry.

References [37, 134, 141–143] compare the utility of a GAN-based to a VAE-based simulator of showers in the standard ATLAS calorimeter geometry, directly relying on shower shapes, effects of calorimeter calibration, reconstruction, and other physics variables

to quantify performance. As opposed to prior work which views calorimeters as volumetric constructs, this work instead unravels the visible calorimeter pixel space, allowing both VAEs and GANs to model interactions using simplified fully-connected layers.

Reference [144] examines the utility of these techniques for an LHCb-like detector modeled in GEANT4. This work models 3D calorimeter showers by opting to sum-reduce along the depth axis, effectively resulting in a single-layer calorimeter. To ensure physical consistency, an auxiliary task is added to the training, requiring a secondary network to reconstruct the momenta and positions p_x , p_y , p_z , x , y , and E_0 . In Fig. 9, the GAN emulator is shown to accurately obey the conditioning that directly controls observations in the simulation.

Reference [145] utilizes the CMS HGCal [146] as a test bed for WGANs when applied to multi-layer calorimeter simulation. In addition to the generator and the critic, [145] utilizes two constraining networks — one for impact position, and one for energy conditioning. As opposed to [132, 133], [145] views the depth dimension as a “channel”, in the strictly computer vision sense, thus utilizing 2D convolutions in lieu of 3D convolutions.

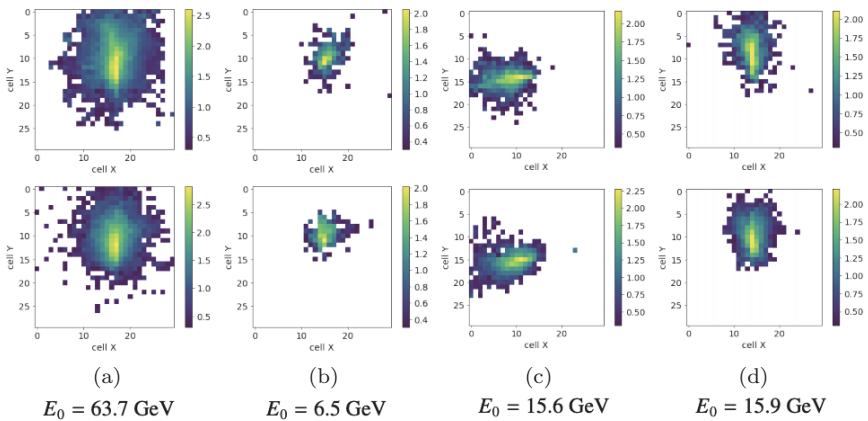


Fig. 9. Showers generated with GEANT4 (top row), and showers simulated by the GAN model (bottom row) for four different sets of initial conditions. Figure reproduced from [144].

As novel representations of irregular detector geometries and particle hits within their volumes are explored, new avenues of research at the intersection of physics and machine learning are likely to bring upon further improvements and increase flexibility in the generative systems designed for fast calorimeter simulation [147].

A common thread across all work building towards generative models as a solution to fast simulation is an emphasis on the fidelity and preservation of physics observables, or shower shapes, in order to verify that domain-specific quantities of interest, such as energy deposition fractions, are well modeled. As a result, many empirical solutions that explicitly encode physical requirements are incorporated in the architecture and objective design; these include energy reconstruction and conditioning requirements [131, 138, 141, 142], sparsity requirements [129], and incident angle consistency [129].

The usage of GANs as a tool for fast simulation within the field has been further validated on new domains, such as the generation of QCD dijet events at the 4-vector level [148] and the generation and subsequent conditioning of Lund jet images [149–152], among others [22–80] (see also [153] for an up-to-date list).

Training a deep generative model directly on data, as opposed to taking GEANT4 simulation as ground truth, is also possible, wherever desirable. The advantage lies in the ability to remove any trace of simulation artifacts and mismodeling that the generative model would inherit from GEANT4. A potential downside is the lack of ground truth conditioning. This option is explored in the GAN-based simulation of the LHCb Ring-Imaging Cherenkov (RICH) detector, which is used for particle identification [154]. This work bypasses the simulation of the high-dimensional low-level detector representation in favor of the direct modeling of a low-dimensional set of high-level reconstructed observables of interest.

As is evident, there exist a myriad of approaches and significant excitement toward the possibility of generative model-based fast simulation of particle showers, replacing, or speeding up, large portions of the traditional simulation pipeline. As a test bed for generative models, and GANs in particular, this problem space offers observable physics quantities, often in the form of shower shapes, that allow

researchers to gauge performance along dimensions of experimental interest and importance. Deep generative models for fast simulation have the potential to model complex, non-parameteric shower distributions and fluctuations, and correlated energy depositions across detector layers — features of electromagnetic and hadronic showers that competing fast simulation techniques do not capture naturally.

Though significant progress has been made towards this ultimate end goal, fundamental questions remain. Each of the works described in this section builds model architectures towards particular geometric assumptions of the target calorimeter. Both geometry- and material-independent generation represent fundamental next steps towards a generative model-powered simulation for the next generation of experiments.

1.5. *Biases in generative modeling*

The evaluation of generative models is plagued by the absence of incontestable quantitative metrics that also meet the necessary condition of appealing to qualitative evaluation standards [155]. The likelihood is not always tractable and feasible to compute. When tasked with the indirect production of natural images, human perception and structural consistency scores have been employed as proxies for success. Examples of common evaluation metrics include the inception score (IS) [97] and the Frechet inception distance (FID) [85]. In scientific applications, researchers often resort to domain-specific loss functions or post-facto evaluation metrics that encode the physical properties of the generated objects that ought to be correctly modelled. High-energy physics is no different, as the work described in Sec. 1.4 usually considers shower shapes or other observables to test modeling fidelity.

Although marginal distributions, divergences, and other quantitative metrics may be available for generative model evaluation, much of the assessment of a method’s potential is still performed via manual (or visual, in the case of computer vision) inspection of the generated samples. This method tends to incur in strong confirmation bias, whenever the observer subconsciously looks for desirable features

and patterns. Furthermore, un-monitored phase space regions of the sample might eventually be used in downstream analysis.

Dataset bias also plays an important role in selective and misleading performance reporting of generative models, in the best case, and in creation of untrustworthy, biased systems, in the worst case. The practice of prototyping generative systems on limited and simplified datasets, with little diversity or unrealistic number of classes or factors of variation, may lead to overestimating the performance of these models in complex, realistic use-cases. While it is natural to begin model development by tackling simpler generative tasks, it is important to avoid premature optimization and over-fitting to these scenarios, and to address the ways in which findings in this regime may not generalize to real world applications. In addition, of particular concern for applications to the physical sciences, any statistical fluctuations in the training set can result systematic biases for the generative model.

Intrinsic inductive biases in a model may also be cause for concern when not carefully evaluated and understood in realistic contexts.

There are other considerations as well, such as the statistical power of a dataset generated with a generative model. In particular, if a generative model is trained with N events and then used to produce $M \gg N$ events, is there any statistical advantage of the M events over the original N ? This topic was explored in [57], where it was demonstrated that when additional information is encoded in the generative model, it is possible to achieve a higher statistical precision than the original dataset. The additional information could simply be in assuming smoothness in a local probability density (as in [57]) or in symmetries of the data themselves.

2. Conclusions and Outlook

Deep generative models are powerful tools that may help to solve a variety of computational challenges in high-energy physics. A variety of studies have shown that such models can capture the salient features of various datasets and therefore may be able to replace or augment existing simulations. Furthermore, some of these tools can

be used directly for inference if they provide a tractable likelihood. Training generative models is significantly harder than training discriminative models because monitoring an entire density is challenging. Achieving the next level of precision for generative models will require innovations on model selection and this is an area that fundamental physics applications may be able to provide unique insight given our detailed first principles simulation models.

References

- [1] Fast simulation of the ATLAS calorimeter system with generative adversarial networks, Technical Report, ATL-SOFT-PUB-2020-006, CERN, Geneva (2020). URL <http://cds.cern.ch/record/2746032>.
- [2] HEP Software Foundation, HEP software foundation community white paper working group — detector simulation (2018).
- [3] J. Schaarschmidt, The new ATLAS fast calorimeter simulation. Technical Report, ATL-SOFT-PROC-2017-005. 4, CERN, Geneva (2017). URL <http://cds.cern.ch/record/2240206>.
- [4] S. Agostinelli *et al.*, Geant4 — a simulation toolkit, *Nucl. Instrum. Methods Phys. Res. Sec. A* **506**(3) (2003) 250–303; doi:10.1016/S0168-9002(03)01368-8.
- [5] A. van den Oord, N. Kalchbrenner and K. Kavukcuoglu, Pixel recurrent neural networks (2016); arXiv:1601.06759.
- [6] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior and K. Kavukcuoglu, Wavenet: A generative model for raw audio (2016); arXiv:1609.03499.
- [7] A. Radford, K. Narasimhan, T. Salimans and I. Sutskever, Improving language understanding by generative pre-training.
- [8] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov and Q. V. Le, Xlnet: Generalized autoregressive pretraining for language understanding (2019); arXiv:1906.08237.
- [9] B. Uria, M.-A. Côté, K. Gregor, I. Murray and H. Larochelle, Neural autoregressive distribution estimation, *J. Mach. Learn. Res.* **17**(1) (2016) 7184–7220.
- [10] M. Germain, K. Gregor, I. Murray and H. Larochelle, Made: Masked autoencoder for distribution estimation, in *Int. Conf. Machine Learning*, (2015) pp. 881–889.
- [11] L. Dinh, J. Sohl-Dickstein and S. Bengio, Density estimation using real NVP (2016); arXiv:1605.08803.
- [12] D. P. Kingma and P. Dhariwal, Glow: Generative flow with invertible 1×1 convolutions (2018); arXiv:1807.03039.
- [13] E. G. Tabak, *et al.*, Density estimation by dual ascent of the log-likelihood, *Commun. Math. Sci.* **8**(1) (2010) 217–233.

- [14] D. J. Rezende and S. Mohamed, Variational inference with normalizing flows (2015); arXiv:1505.05770.
- [15] A. Odena, Open questions about generative adversarial networks, *Distill* (2019); doi:10.23915/distill.00018. <https://distill.pub/2019/gan-open-problems>.
- [16] D. P. Kingma and M. Welling, Auto-encoding variational bayes, in *Int. Conf. Learning Representations (ICLR)* (2014).
- [17] I. Goodfellow *et al.*, Generative adversarial nets, in *Advances in Neural Information Processing Systems* (2014) pp. 2672–2680.
- [18] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf and A. Smola, A kernel two-sample test, *J. Mach. Learn. Res.* **13**(1) (2012) 723–773.
- [19] G. K. Dziugaite, D. M. Roy and Z. Ghahramani, Training generative neural networks via maximum mean discrepancy optimization (2015); arXiv:1505.03906.
- [20] C.-L. Li, W.-C. Chang, Y. Cheng, Y. Yang and B. Póczos, MMD GAN: Towards deeper understanding of moment matching network, in *Advances in Neural Information Processing Systems* (2017) pp. 2203–2213.
- [21] Y. Song and S. Ermon, Generative modeling by estimating gradients of the data distribution (2019).
- [22] L. de Oliveira, M. Paganini and B. Nachman, Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis, *Comput. Softw. Big Sci.* **1** (2017) 4.
- [23] M. Paganini, L. de Oliveira and B. Nachman, Accelerating science with generative adversarial networks: An application to 3d particle showers in multilayer calorimeters, *Phys. Rev. Lett.* **120**(4) (2018) 042003; doi:10.1103/PhysRevLett.120.042003.
- [24] M. Paganini, L. de Oliveira and B. Nachman, CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks, *Phys. Rev. D* **97**(1) (2018) 014021; doi:10.1103/PhysRevD.97.014021.
- [25] S. Alonso-Monsalve and L. H. Whitehead, Image-based model parameter optimization using model-assisted generative adversarial networks (2018); doi:10.1109/TNNLS.2020.2969327.
- [26] A. Butter, T. Plehn and R. Winterhalder, How to GAN event subtraction, *SciPost Phys. Core* **3** (2020) 009.
- [27] J. Arjona Martinez, T. Q. Nguyen, M. Pierini, M. Spiropulu and J.-R. Vlimant, Particle generative adversarial networks for full-event simulation at the LHC and their application to pileup description, *J. Phys. Conf. Ser.* **1525** (2020) 012081.
- [28] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn and R. Winterhalder, How to GAN away detector effects, *SciPost Phys.* **8** (2020) 070.
- [29] S. Vallecorsa, F. Carminati and G. Khattak, 3D convolutional GAN for fast simulation, in *Proc. 23rd Int. Conf. Computing in High Energy and Nuclear Physics (CHEP 2018)*, Vol. 214, Sofia, Bulgaria, July 9–13, 2018. (2019); doi:10.1051/epjconf/201921402010.

- [30] C. Ahdida *et al.*, Fast simulation of muons produced at the SHiP experiment using generative adversarial networks, *J. Instrum.* **14** (2019) P11028.
- [31] S. Carrazza and F. A. Dreyer, Lund jet images from generative and cycle-consistent adversarial networks, *Eur. Phys. J. C* **79**(11) (2019) 979; doi:10.1140/epjc/s10052-019-7501-1.
- [32] A. Butter, T. Plehn and R. Winterhalder, How to GAN LHC events, *SciPost Phys.* **7** (2019) 075; doi:10.21468/SciPostPhys.7.6.075.
- [33] J. Lin, W. Bhimji and B. Nachman, Machine learning templates for QCD factorization in the search for physics beyond the standard model, *J. High Energy Phys.* **05** (2019) 181; doi:10.1007/JHEP05(2019)181.
- [34] R. Di Sipio, M. Fauci Giannelli, S. Ketabchi Haghighat and S. Palazzo, DijetGAN: A generative-adversarial network approach for the simulation of QCD dijet events at the LHC, *J. High Energy Phys.* **2019** (2019) 110.
- [35] B. Hashemi, N. Amin, K. Datta, D. Olivito and M. Pierini, LHC analysis-specific datasets with generative adversarial networks (2019); arXiv:1901.05282.
- [36] V. Chekalina, E. Orlova, F. Ratnikov, D. Ulyanov, A. Ustyuzhanin and E. Zakharov, Generative models for fast calorimeter simulation. LHCb case, *EPJ Web Conf.* **214** (2019) 02034.
- [37] Deep generative models for fast shower simulation in ATLAS, Technical Report, ATL-SOFT-PUB-2018-001, CERN, Geneva (2018). URL <https://cds.cern.ch/record/2630433>.
- [38] K. Zhou, G. Endrodi, L.-G. Pang and H. Stocker, Regressive and generative neural networks for scalar field theory, *Phys. Rev. D* **100**(1) (2019), 011501; doi:10.1103/PhysRevD.100.011501.
- [39] F. Carminati, A. Gheata, G. Khattak, P. Mendez Lorenzo, S. Sharan and S. Vallecorsa, Three dimensional Generative Adversarial Networks for fast simulation, in *Proc. 18th Int. Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017)*, Vol. 1085(3), Seattle, WA, USA, August 21–25, 2017, p. 032016 (2018); doi:10.1088/1742-6596/1085/3/032016.
- [40] S. Vallecorsa, Generative models for fast simulation, in *Proc. 18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017)*, Seattle, WA, USA, August 21–25, 2017. Vol. 1085(2), p. 022005 (2018); doi:10.1088/1742-6596/1085/2/022005.
- [41] K. Datta, D. Kar and D. Roy, Unfolding with generative adversarial networks (2018); arXiv:1806.00433.
- [42] P. Musella and F. Pandolfi, Fast and accurate simulation of particle detectors using generative adversarial networks, *Comput. Softw. Big Sci.* **2**(1) (2018) 8; doi:10.1007/s41781-018-0015-y.
- [43] M. Erdmann, L. Geiger, J. Glombitza and D. Schmidt, Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks, *Comput. Softw. Big Sci.* **2**(1) (2018) 4; doi:10.1007/s41781-018-0008-x.
- [44] K. Deja, T. Trzcinski and u. Graczykowski, Generative models for fast cluster simulations in the TPC for the ALICE experiment, in *Proc. 23rd*

- Int. Conf. Computing in High Energy and Nuclear Physics (CHEP 2018)*, Vol. 214, Sofia, Bulgaria, July 9–13, 2018. p. 06003 (2019); doi:10.1051/epjconf/201921406003.
- [45] D. Derkach, N. Kazeev, F. Ratnikov, A. Ustyuzhanin and A. Volokhova, Cherenkov detectors fast simulation using neural networks (2019); doi:10.1016/j.nima.2019.01.031.
 - [46] H. Erbin and S. Krippendorf, GANs for generating EFT models, *Phys. Lett. B* **810** (2020) 135798.
 - [47] M. Erdmann, J. Glombitza and T. Quast, Precise simulation of electromagnetic calorimeter showers using a Wasserstein generative adversarial network, *Comput. Softw. Big Sci.* **3**(1) (2019) 4; doi:10.1007/s41781-018-0019-7.
 - [48] J. M. Urban and J. M. Pawlowski, Reducing Autocorrelation Times in Lattice Simulations with Generative Adversarial Networks, *Mach. Learn. Sci. Technol.* **1**(4) (2020) 045011.
 - [49] L. de Oliveira, M. Paganini and B. Nachman, Tips and tricks for training GANs with physics constraints (2017). URL https://dl4physicsciences.github.io/files/nips_dlps_2017_26.pdf.
 - [50] L. de Oliveira, M. Paganini and B. Nachman, Controlling physical attributes in GAN-accelerated simulation of electromagnetic calorimeters, *J. Phys. Conf. Ser.* **1085**(4) (2018) 042017; doi:10.1088/1742-6596/1085/4/042017.
 - [51] S. Farrell, W. Bhimji, T. Kurth, M. Mustafa, D. Bard, Z. Lukic, B. Nachman and H. Patton, Next generation generative neural networks for HEP, *EPJ Web Conf.* **214** (2019) 09005; doi:10.1051/epjconf/201921409005.
 - [52] B. Hooberman, A. Farbin, G. Khattak, V. Pacela, M. Pierini, J.-R. Vlimant, M. Spiropulu, W. Wei, M. Zhang and S. Vallecorsa, Calorimetry with Deep learning: Particle classification, energy regression, and simulation for high-energy physics (2017). URL https://dl4physicsciences.github.io/files/nips_dlps_2017_15.pdf.
 - [53] D. Belayneh *et al.*, Calorimetry with deep learning: Particle simulation and reconstruction for collider physics, *Eur. Phys. J. C* **80** (2020) 688.
 - [54] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol and K. Krüger, Getting high: High fidelity simulation of high granularity calorimeters with high speed, *Comput. Softw. Big Sci.* **5** (2021) 13.
 - [55] Y. Alanazi *et al.*, AI-based Monte Carlo event generator for electron-proton scattering (2020); arXiv:2008.03151.
 - [56] S. Diefenbacher, E. Eren, G. Kasieczka, A. Korol, B. Nachman and D. Shih, DCTRGAN: Improving the precision of generative models with reweighting, *J. Instrum.* **15** (2020) P11004; doi:10.1088/1748-0221/15/1/p11004.
 - [57] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman and T. Plehn, GANplifying event samples, *SciPost Phys.* **10** (2021) 139.
 - [58] R. Kansal, J. Duarte, B. Orzari, T. Tomei, M. Pierini, M. Touranakou, J.-R. Vlimant and D. Gunopoulos, Graph generative adversarial networks

- for sparse data generation in high energy physics, in *34th Conf. Neural Information Processing Systems* (2020).
- [59] J. W. Monk, Deep learning as a parton shower, *J. High Energ. Phys.* **12** (2018) 021.
 - [60] T. Cheng, J.-F. Arguin, J. Leissner-Martin, J. Pilette and T. Golling, Variational autoencoders for anomalous jet tagging (2020); arXiv:2007.01850.
 - [61] K. Dohi, Variational autoencoders for jet simulation (2020); arXiv:2009.04842.
 - [62] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, JUNIPR: A framework for unsupervised machine learning in particle physics, *Eur. Phys. J. C* **79** (2019) 12.
 - [63] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, Binary JUNIPR: An interpretable probabilistic model for discrimination, *Phys. Rev. Lett.* **123**(18) (2019) 182001; doi:10.1103/PhysRevLett.123.182001.
 - [64] M. Jercic and N. Poljak, Exploring the possibility of a recovery of physics process properties from a neural network model, *Entropy* **22**(9) (2020) 994.
 - [65] M. S. Albergo, G. Kanwar and P. E. Shanahan, Flow-based generative models for Markov chain Monte Carlo in lattice field theory, *Phys. Rev. D* **100**(3) (2019) 034515; doi:10.1103/PhysRevD.100.034515.
 - [66] G. Kanwar, M. S. Albergo, D. Boyda, K. Cranmer, D. C. Hackett, S. Racanière, D. J. Rezende and P. E. Shanahan, Equivariant flow-based sampling for lattice gauge theory, *Phys. Rev. Lett.* **125** (2020) 121601.
 - [67] J. Brehmer and K. Cranmer, Flows for simultaneous manifold learning and density estimation (2020); arXiv:2003.13913.
 - [68] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale, and S. Schumann, Exploring phase space with neural importance sampling, *SciPost Phys.* **8** (2020) 069.
 - [69] C. Gao, S. Höche, J. Isaacson, C. Krause and H. Schulz, Event generation with normalizing flows, *Phys. Rev. D.* **101**(7) (2020) 076002; doi:10.1103/PhysRevD.101.076002.
 - [70] C. Gao, J. Isaacson and C. Krause, i-flow: High-dimensional integration and sampling with normalizing flows, *Mach. Learn. Sci. Technol.* **1** (2020) 045023.
 - [71] B. Nachman and D. Shih, Anomaly Detection with density estimation, *Phys. Rev. D.* **101** (2020) 075042; doi:10.1103/PhysRevD.101.075042.
 - [72] S. Choi, J. Lim and H. Oh, Data-driven estimation of background distribution through neural autoregressive flows (2020); arXiv:2008.03636.
 - [73] Y. Lu, J. Collado, D. Whiteson and P. Baldi, SARM: Sparse autoregressive model for scalable generation of sparse images in particle physics, *Phys. Rev. D* **103** (2021) 036012.
 - [74] C. Chen, O. Cerri, T. Q. Nguyen, J.-R. Vlimant and M. Pierini, Data augmentation at the LHC through analysis-specific fast simulation with deep learning (2020); arXiv:2010.01835.
 - [75] J. Bendavid, Efficient Monte Carlo integration using boosted decision trees and generative deep neural networks (2017); arXiv:1707.00028.

- [76] M. D. Klimek and M. Perelstein, Neural network-based approach to phase space integration, *SciPost Phys.* **9** (2020) 053.
- [77] S. Carrazza and J. M. Cruz-Martinez, VegasFlow: accelerating Monte Carlo simulation across multiple hardware platforms, *Comput. Phys. Commun.* **254** (2020) 107376.
- [78] B. Nachman and J. Thaler, A neural resampler for Monte Carlo reweighting with preserved uncertainties, *Phys. Rev. D* **102** (2020) 076004.
- [79] I.-K. Chen, M. D. Klimek and M. Perelstein, Improved neural network Monte Carlo simulation, *SciPost Phys.* **10** (2021) 023.
- [80] R. Verheyen and B. Stienen, Phase space sampling and inference from weighted events with autoregressive flows, *SciPost Phys.* **10** (2021) 038.
- [81] M. Mustafa, D. Bard, W. Bhimji, Z. Lukić, R. Al-Rfou and J. M. Kratochvil, Cosmogon: Creating high-fidelity weak lensing convergence maps using generative adversarial networks, *Comput. Astrophys. Cosmology* **6**(1) (2019) 1.
- [82] A. Osokin, A. Chessel, R. E. Carazo Salas and F. Vaggi, Gans for biological image synthesis, in *The IEEE Int. Conf. Computer Vision (ICCV)* (Oct, 2017).
- [83] H. Jiang, Z. Nie, R. Yeo, A. B. Farimani and L. B. Kara, Stressgan: A generative deep learning model for 2d stress distribution prediction, *J. Appl. Mech.* **88**(5) (2021) 051005.
- [84] O. Hennigh, Lat-net: Compressing lattice boltzmann flow simulations using deep neural networks (2017); arXiv:1705.09036.
- [85] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, GANs trained by a two time-scale update rule converge to a local nash equilibrium (2017); arXiv:1706.08500.
- [86] J. Li, A. Madry, J. Peebles and L. Schmidt, On the limitations of first-order approximation in GAN dynamics (2017); arXiv:1706.09884.
- [87] V. Nagarajan and J. Z. Kolter, Gradient descent GAN optimization is locally stable (2017); arXiv:1706.04156.
- [88] L. Mescheder, S. Nowozin and A. Geiger, The numerics of GANs (2017); arXiv:1705.10461.
- [89] L. Mescheder, A. Geiger and S. Nowozin. Which training methods for GANs do actually converge? (2018); arXiv:1801.04406.
- [90] H. Berard, G. Gidel, A. Almahairi, P. Vincent and S. Lacoste-Julien, A closer look at the optimization landscapes of generative adversarial networks (2019); arXiv:1906.04848.
- [91] M. Lucic, K. Kurach, M. Michalski, S. Gelly and O. Bousquet, Are gans created equal? a large-scale study, in *Advances in Neural Information Processing Systems* (2018), pp. 700–709.
- [92] A. Radford, L. Metz and S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks (2015); arXiv:1511.06434.
- [93] T. Karras, T. Aila, S. Laine and J. Lehtinen, Progressive growing of gans for improved quality, stability, and variation (2017); arXiv:1710.10196.

- [94] A. Odena, C. Olah and J. Shlens, Conditional image synthesis with auxiliary classifier GANs, in *Proc. 34th Int. Conf. Machine Learning*, Vol. 70 (2017) pp. 2642–2651.
- [95] M. Arjovsky, S. Chintala and L. Bottou, Wasserstein generative adversarial networks, in *Proc. 34th Int. Conf. Machine Learning*, Vol. 70 (2017), pp. 214–223.
- [96] I. Gulrajani et al., Improved training of wasserstein gans. in *Advances in Neural Information Processing Systems* (2017), pp. 5767–5777.
- [97] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, Improved techniques for training gans, in *Advances in Neural Information Processing Systems* (2016), pp. 2234–2242.
- [98] B. K. Sriperumbudur, K. Fukumizu, A. Gretton, B. Schölkopf and G. R. G. Lanckriet, On integral probability metrics, divergences and binary classification (2009); arXiv:0901.2698.
- [99] Y. Li, A. Schwing, K.-C. Wang and R. Zemel, Dualing GANs (2017); arXiv:1706.06216.
- [100] A. B. Dieng, F. J. R. Ruiz, D. M. Blei, and M. K. Titsias, Prescribed generative adversarial networks (2019); arXiv:1910.04302.
- [101] C. Daskalakis, A. Ilyas, V. Syrgkanis and H. Zeng, Training GANs with optimism (2017); arXiv:1711.00141.
- [102] A. Brock, J. Donahue and K. Simonyan, Large scale gan training for high fidelity natural image synthesis (2018); arXiv:1809.11096.
- [103] C. K. Sønderby, J. Caballero, L. Theis, W. Shi and F. Huszár, Amortised map inference for image super-resolution (2016); arXiv:1610.04490.
- [104] C. Wu, L. Herranz, X. Liu, Y. Wang, J. van de Weijer and B. Raducanu, Memory replay GANs: learning to generate images from new categories without forgetting (2018); arXiv:1809.02058.
- [105] K. Liu, W. Tang, F. Zhou and G. Qiu, Spectral regularization for combating mode collapse in GANs (2019); arXiv:1908.10999.
- [106] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann and C. Sutton, Vee-gan: Reducing mode collapse in GANs using implicit variational learning, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (Curran Associates, Inc., 2017), pp. 3308–3318.
- [107] C. Villani, *Optimal Transport: Old and New*, Vol. 338 (Springer Science & Business Media, 2008).
- [108] T. Miyato, T. Kataoka, M. Koyama and Y. Yoshida, Spectral normalization for generative adversarial networks, preprint (2018); arXiv:1802.05957.
- [109] S. Arora, R. Ge, Y. Liang, T. Ma and Y. Zhang, Generalization and equilibrium in generative adversarial nets (GANs) (2017); arXiv:1703.00573.
- [110] C. K. Fisher, A. M. Smith and J. R. Walsh, Boltzmann encoded adversarial machines (2018); arXiv:1804.08682.
- [111] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow and B. Frey, Adversarial autoencoders (2015); arXiv:1511.05644.

- [112] D. M. Blei, A. Kucukelbir and J. D. McAuliffe, Variational inference: A review for statisticians, *J. Amer. Statist. Assoc.* **112**(518) (2017) 859–877.
- [113] J. de Favereau *et al.*, DELPHES 3: A modular framework for fast simulation of a generic collider experiment, *J. High Energy Phys.* **2014**(2) (2014); doi:10.1007/jhep02(2014)057.
- [114] E. Barberio, *et al.*, Fast simulation of electromagnetic showers in the atlas calorimeter: Frozen showers, *J. Phys. Conf. Ser.* **160** (2009) 012082.
- [115] G. Grindhammer, M. Rudowicz and S. Peters, The fast simulation of electromagnetic and hadronic showers, *Nucl. Instrum. Methods Phys. Res. Sect. A.* **290**(2) (1990) 469–488; doi:10.1016/0168-9002(90)90566-O.
- [116] S. J. Gasiorowski, Fast simulation in ATLAS, Technical Report, ATL-SOFT-PROC-2020-027, CERN, Geneva (2020). URL <http://cds.cern.ch/record/2712930>.
- [117] S. Sekmen, Recent developments in CMS fast simulation (2017); arXiv:1701.03850.
- [118] S. Abdullin *et al.*, The fast simulation of the CMS detector at LHC, *J. Phys. Conf. Ser.* **331**(3) (2011) 032049; doi:10.1088/1742-6596/331/3/032049.
- [119] The new fast calorimeter simulation in ATLAS, Technical Report, ATL-SOFT-PUB-2018-002, CERN, Geneva (2018). URL <https://cds.cern.ch/record/2630434>.
- [120] K. Edmonds, S. Fleischmann, T. Lenz, C. Magass, J. Mechnich and A. Salzburger. The fast ATLAS track simulation (FATRAS), Technical Report, ATL-SOFT-PUB-2008-001. ATL-COM-SOFT-2008-002, CERN, Geneva (2008). URL <http://cds.cern.ch/record/1091969>.
- [121] C. ATLAS, M. Beckingham, M. Duehrssen, E. Schmidt, M. Shapiro, M. Venturi, J. Virzi, I. Vivarelli, M. Werner, S. Yamamoto and T. Yamanaka, The simulation principle and performance of the ATLAS fast calorimeter simulation FastCaloSim, Technical Report, ATL-PHYS-PUB-2010-013, CERN, Geneva (2010). URL <http://cds.cern.ch/record/1300517>.
- [122] Performance of the fast ATLAS tracking simulation (FATRAS) and the ATLAS fast calorimeter simulation (FastCaloSim) with single particles, Technical Report, ATL-SOFT-PUB-2014-001, CERN, Geneva (2014). URL <https://cds.cern.ch/record/1669341>.
- [123] L. de Oliveira, M. Paganini and B. Nachman, Learning particle physics by example: location-aware generative adversarial networks for physics synthesis, *Comput. Softw. Big Sci.* **1**(1) (2017) 4.
- [124] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman and A. Schwartzman, Jet-images — deep learning edition, *J. High Energy Phys.* **2016**(7) (2016) 69.
- [125] T. Sjöstrand, S. Mrenna and P. Skands, A brief introduction to PYTHIA 8.1, *Comput. Phys. Commun.* **178**(11) (2008) 852–867; doi:10.1016/j.cpc.2008.01.036.
- [126] L. de Oliveira and M. Paganini, lukedeo/adversarial-jets: Initial release (2017); doi:10.5281/zenodo.400708.

- [127] B. Nachman, L. de Oliveira and M. Paganini, Dataset release — Pythia generated jet images for location aware generative adversarial network training (2017); doi:10.17632/4r4v785rgx.1.
- [128] M. Erdmann *et al.*, Generating and refining particle detector simulations using the wasserstein distance in adversarial networks, *Comput. Softw. Big Sci.* **2**(1) (2018); doi:10.1007/s41781-018-0008-x.
- [129] L. de Oliveira, M. Paganini and B. Nachman, Controlling physical attributes in gan-accelerated simulation of electromagnetic calorimeters *J. Phys. Conf. Ser.* **1085** (2018) 042017.
- [130] M. Paganini, L. de Oliveira and B. Nachman, CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks, *Phys. Rev. D* **97**(1) (2018) 014021.
- [131] M. Paganini, L. de Oliveira and B. Nachman, Accelerating science with generative adversarial networks: an application to 3D particle showers in multilayer calorimeters, *Phys. Rev. Lett.* **120**(4) (2018) 042003.
- [132] F. Carminati, G. Khattak and S. Vallecorsa, 3D convolutional GAN for fast simulation, in *23rd Int. Conf. Computing High Energy and Nuclear Physics (CHEP 2018)* (2019).
- [133] D. Belayneh *et al.*, Calorimetry with deep learning: Particle simulation and reconstruction for collider physics (2019); arXiv:1912.06794.
- [134] K. Cranmer *et al.*, Deep generative models for fast shower simulation in ATLAS, in *Third Workshop on Bayesian Deep Learning, NeurIPS 2018* (2018)
- [135] M. Paganini, L. de Oliveira, and B. Nachman, hep-lbdl/CaloGAN: CaloGAN generation, training, and analysis code (2017); doi:10.5281/zenodo.584155.
- [136] B. Nachman, L. de Oliveira and M. Paganini, Dataset release — electromagnetic calorimeter shower images (2017); doi:10.17632/pvn3xc3wy5.1.
- [137] P. Lebrun, L. Linssen, A. Lucaci-Timoce, D. Schulte, F. Simon, S. Stapnes, N. Toge, H. Weerts and J. Wells, The CLIC programme: Towards a staged e+e- linear collider exploring the terascale : CLIC conceptual design report (2012); arXiv:1209.2543.
- [138] S. Voloshynovskiy, M. Kondah, S. Rezaeifar, O. Taran, T. Holotyak and D. J. Rezende, Information bottleneck through variational glasses (2019); arXiv:1912.00830.
- [139] T. I. Collaboration. International large detector: Interim design report (2020); arXiv:2003.01116.
- [140] A. Butter, T. Plehn and R. Winterhalder, How to GAN LHC events, *SciPost Phys.* **7** (2019) 075.
- [141] Double critic GAN for fast shower simulation in ATLAS, Technical Report, ATL-SOFT-SIM-2019-004, CERN, Geneva (2019). URL <https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/SIM-2019-004/>.
- [142] ATLAS fast calorimeter simulation validation for NEC, Technical Report, ATL-SOFT-SIM-2019-006, CERN, Geneva (2019). URL <https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/SIM-2019-006/>.

- [143] Improved VAE for fast shower simulation in ATLAS, Technical Report, ATL-SOFT-SIM-2019-007, CERN, Geneva (2019). URL <https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/SIM-2019-007/>.
- [144] V. Chekalina *et al.*, Generative models for fast calorimeter simulation: the LHCb case, *EPJ Web Conf.* **214** (2019) 02034; doi:10.1051/epjconf/201921402034.
- [145] M. Erdmann, J. Glombitza and T. Quast, Precise simulation of electromagnetic calorimeter showers using a Wasserstein generative adversarial network, *Comput. Softw. Big Sci.* **3**(1) (2019); doi:10.1007/s41781-018-0019-7.
- [146] A. Martelli, The CMS HGCal detector for HL-LHC upgrade (2017); arXiv:1708.08234.
- [147] S. R. Qasim *et al.*, Learning representations of irregular particle-detector geometry with distance-weighted graph networks, *Eur. Phys. J. C.* **79**(7) (2019); doi:10.1140/epjc/s10052-019-7113-9.
- [148] R. Di Sipio *et al.*, DijetGAN: A generative-adversarial network approach for the simulation of QCD dijet events at the LHC, *J. High Energy Phys.* **2019**(8) (2019); doi:10.1007/jhep08(2019)110.
- [149] S. Carrazza and F. Dreyer, Lund jet images from generative and cycle-consistent adversarial networks, *Eur. Phys. J. C* **79**(11) (2019); doi:10.1140/epjc/s10052-019-7501-1.
- [150] F. Dreyer and S. Carrazza, JetsGame/CycleJet v1.0.0 (2019); doi:10.5281/zenodo.3384919.
- [151] F. Dreyer and S. Carrazza, JetsGame/gLund v1.0.0 (2019); doi:10.5281/zenodo.3384921.
- [152] S. Carrazza and F. Dreyer, JetsGame/data v1.0.0 (2019); doi:10.5281/zenodo.2602515.
- [153] HEP ML Community, A living review of machine learning for particle physics, URL <https://iml-wg.github.io/HEPML-LivingReview/>.
- [154] A. Maevskiy, D. Derkach, N. Kazeev, A. Ustyuzhanin, M. Artemev, and L. Anderlini. Fast data-driven simulation of cherenkov detectors using generative adversarial networks, in *19th Int. Workshop on Advanced Computing and Analysis Techniques in Physics Research: Empowering the revolution: Bringing Machine Learning to High Performance Computing* (2019).
- [155] L. Theis, A. van den Oord and M. Bethge, A note on the evaluation of generative models (2015); arXiv:1511.01844.

This page intentionally left blank

Chapter 7

Generative Networks for LHC Events

Anja Butter* and Tilman Plehn

*Institut für Theoretische Physik
Universität Heidelberg
Germany*

**butter@thphys.uni-heidelberg.de*

LHC physics crucially relies on our ability to simulate events efficiently from first principles. Modern machine learning, specifically generative networks, will help us tackle simulation challenges for the coming LHC runs. Such networks can be employed within established simulation tools or as part of a new framework. Since some neural network architectures can be inverted, they also open new avenues in LHC analyses.

1. Introduction

Machine learning in particle physics has a set of defining features, as compared to many other applications of similar techniques. First, LHC produces proper *big data*, which means that for instance in QCD we typically work with many millions of jets as analysis objects. Second, LHC events in all their complexity are described by simple laws of *fundamental physics*, formulated in terms of perturbative quantum field theory. Third, these predictions are available through *first-principles simulations* for the hard scattering process and the non-perturbative QCD effects, all the way to detector simulations [1–4]. Finally, these simulation tools are publicly available and can be combined with public datasets to develop meaningful new analysis ideas even from outside the experimental collaborations.

1.1. *Machine learning opportunities*

In this review, we focus on machine learning approaches to improving LHC event simulations. Currently, all these simulations are based on Monte Carlo methods. Specific advantages of generative networks over Monte Carlo simulations include (i) the fact that they are extremely fast once trained, (ii) that they can be trained on any combination of simulated and actual data, and (iii) that they can be constructed in an easily invertible manner.

Simulations are not only an obvious machine learning application at the LHC, they are also the basis of many other envisioned machine learning improvements. Following a straightforward big-data approach we can, for instance, attempt to simulate a full Run 3 or HL-LHC dataset, and then compare simulated and observed data at an event-to-event level. This is the idea behind the planned simulation-based or likelihood-free, so-called legacy analyses introduced in Chapter 16. It is important to recognize that this implicitly assumes that we can simulate such a dataset from first principles with sufficient precision. More than anything these new inference methods rely on conceptual progress in fast precision simulations. For our typical statistical approaches such analyses also assume a set of established signal hypotheses, while in practice we constantly modify theory frameworks and adapt them to detector and background challenges or hints of new physics. For our first-principles simulations this means that we have to provide not only precise and fast, but also flexible signal simulations.

The development and validation of fast precision simulations requires a close collaboration between theory and experiment. This is why we view the comparison of simulated and measured LHC events as a dynamic system, where theory and experiment develop their respective tools in a constant exchange. Here it is helpful to understand simulations as a chain of independent steps. They start with the hard scattering described by perturbative quantum field theory in the form of a Lagrangian. Jet radiation and parton showers are described by resummed perturbative quantum field theory. Next comes hadronization and fragmentation, and finally a detector

simulation which allows us to compare the result to the 4-momenta of identified particles in the detectors. Each of these modules requires a continuous improvement in our understanding of the data, the precision of the theoretical calculations, and often the tuning of a minimal number of physically plausible tuning parameters.

With these specific machine learning applications in mind, we split this review into four physics sections. In Sec. 2, we briefly review the kind of neural networks which are used in LHC simulations. Our focus will be on generative networks, but we will mention some other applications in passing. Next, we discuss different ways deep networks are used for specific event generation tasks in in Sec. 3. These tasks reflect the modular nature of LHC simulations, and the network architecture as well as the training data format are adapted to the respective physics task. As an alternative, we discuss generative networks trained on full events in Sec. 4. The output of these networks can be parton-level events or events after a fast detector simulation, and we will omit a detailed discussion of detector simulation because this is discussed in Chapter 6. Finally, we introduce conceptual and practical opportunities from inverting the LHC simulation chain in Sec. 5.

1.2. *LHC motivation*

The upcoming LHC runs pose a serious challenge to theory predictions and simulations [5]. By the end of the HL-LHC run we hope to analyze of the order of 25 times the number of relevant events as we have available after Run 2. This estimate follows directly from the increase in energy and luminosity, and for the bulk of phase space it also translates into a factor five smaller statistical uncertainties on cross-section measurements. This means that typical current statistical, systematic, and theory uncertainties in the range of 5–10% have to be reduced to the per-cent level. First of all, this includes the statistical limitation through the number of simulated events, where the size of the simulated sample should grow with the expected dataset.

In general, an improvement in simulation speed is equivalent to an improvement in precision, assuming that the required theory

predictions are available in an appropriate form. From a perturbative QCD or electroweak point of view, reducing the theory uncertainty by a factor five means going to the next order. For typical LHC simulations this implies a shift from standard LO-NLO level to NLO-NNLO, including kinematic distributions. Such an increase in perturbative order includes one loop order more for virtual corrections and one more final-state particle for real corrections, both aspects corresponding to an increase in complexity by at least an order of magnitude. Closely related to the perturbative expansion, more detailed kinematic analyses require more precise simulations especially in exotic, high-multiplicity phase space regions. An example is the jet recoil associated with any kind of hard process, again leading to a significant increase in simulation complexity.

Standard LHC analyses do not simulate all possible signal and background channels, but focus on those which are expected to contribute after a set of basic kinematic cuts and given the available luminosity. With higher luminosities and higher precision, we will become sensitive to additional backgrounds. Such background channels are often estimated using Monte Carlo simulations and typically include high-multiplicity final states and tails of kinematic distributions, challenging the simulation framework.

Finally, the experimental uncertainties on rate measurements in the bulk of phase space are becoming very small already at Run 2. At the HL-LHC level it is not clear how we can include the corresponding high-loop predictions in the standard simulation chain, even for key processes. While serious effort has been going into translating fiducial measurements into a predicted total cross-section, it is not clear how this approach can be generalized to simple kinematic distributions for the hard scattering process. On the experimental side, it is not always clear what kind of signal hypothesis should be used to report a measurement, for instance in the case of topologically defined searches for physics beyond the Standard Model. In addition to extracting the best possible limits for certain models we would like to unfold kinematic structures or the detector simulation. From a simulation perspective the last two tasks are linked to a simple question, namely how to invert the event simulation.

A third application of this technical task is likelihood-based inference through the so-called matrix element method.

This leads to a short, likely incomplete list of goals for theory simulations for the future LHC runs, specifically the HL-LHC, namely

- simulated event numbers scaling with the expected events;
- general move to NLO/NNLO as standard precision;
- higher relevant final-state multiplicities;
- additional low-rate high-multiplicity backgrounds;
- specific precision predictions not available in standard generators;
- interpretation of measurements without a signal hypothesis;

To illustrate the available computing resources, Fig. 1 estimates the annual CPU computing needs by ATLAS until 2034 [6]. The baseline R&D scenario lies far outside the available resources assuming computing power increases by 10% to 20% per year. With a well-defined aggressive R&D effort, it should be possible to meet the computing needs of future LHC runs. In the right panel of Fig. 1, we see that event generation is not the only CPU-intensive aspect of LHC physics, but it is expected to contribute close to 20% to the budget. All tasks combined define the CPU usage in the left panel. If any of them leads to a significant increase in simulation time, this task will become a limiting factor in Run 3 or HL-LHC analyses. There exist

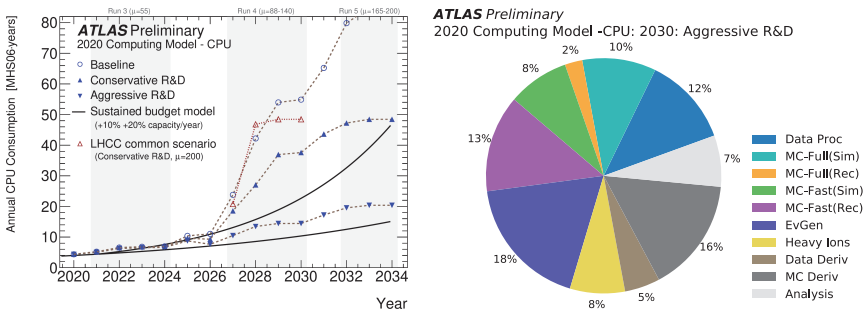


Fig. 1. Left: estimated CPU resources needed per year. The solid lines indicate annual improvements of 10% and 20% in new hardware. The blue symbols represent three scenarios following the current LHC schedule. Right: projected CPU usage in 2030 for the aggressive R&D scenario. Figures taken from [6].

already Run-2 analyses for which theory simulations are the limiting factor in extracting, for instance, Higgs properties. Unless we can significantly improve first-principle precision simulations, this will become a standard problem for HL-LHC analyses. This defines the main task of the LHC theory and simulation community, and on the simulation side machine learning should help us meet our precision goals.

2. Generative Networks

Generative networks are machine learning tools which generate new samples following a learned distribution. The generated data can have the same form as the training data, in which case the generative network will produce statistically independent samples reproducing the implicit underlying structures of the training data. While there are several types of generative networks available, we will focus on models that have been applied successfully to LHC event generation: generative adversarial neural networks (GANs), variational autoencoders (VAEs), and normalizing flows (NFs). The input of a generative network may in principle depend on conditional parameters, however we start by considering unconditional generative networks and keep in mind that they can always be extended to include conditional information.

2.1. Generative adversarial networks

The standard generative adversarial network consists of two networks, a generator G and a discriminator D acting as adversaries. The discriminator is trained to distinguish samples of the generated distribution P_G from samples of the true data distribution P_T . The last layer of the discriminator maps its output to the range $D \in [0, 1]$. Minimizing the loss function

$$L_D = \langle -\log D(x) \rangle_{x \sim P_T} + \langle -\log(1 - D(x)) \rangle_{x \sim P_G} \quad (1)$$

tags true events with the label $D = 1$ and generated events with the label $D = 0$. The brackets $\langle \cdot \rangle_{x \sim P}$ indicate the expectation value with respect to the distribution P . In the next step, the generator adjusts

the generated samples by minimizing its loss function

$$L_G = \langle -\log D(x) \rangle_{x \sim P_G}. \quad (2)$$

It pushes the discriminator label of the generated events closer to $D = 1$, marking true events. The combined training alternates the minimization of both loss functions and yields generated event samples following the distribution of the data. An important advantage of the GAN setup is the ability to generate particular realistic samples. However, GANs have a tendency to suffer from unstable training, preventing the convergence to a well-performing minimum. These stability issues can be addressed by adjusting the loss function or adding regularization terms.

Instabilities of the training are often linked to problems in following the gradient of the loss function. Diverging gradients for the discriminator lead to strong oscillations in the loss function, preventing a stable convergence. This can be avoided by adding a regularization term to the discriminator loss that punishes large gradient values [7]. Vanishing gradients, on the other hand, lead to infinitesimal updates of the weights and hence very inefficient training. This problem typically arises when the discriminator is too powerful and easily distinguishes between true and generated events. The logarithmic loss function then leads to zero gradients. The Least Square GAN (LSGAN) solves this problem by replacing the loss function with a squared term [8].

A popular approach to improving GAN training are Wasserstein GANs [9]. While the vanilla GAN minimizes the Jensen–Shannon divergence, the WGAN minimizes the Wasserstein or Earth Mover (EM) distance between the distributions P_T and P_G . The Wasserstein distance is given by

$$W(P_T, P_G) = \inf_{\gamma \in \Pi(P_T, P_G)} \langle \|x - y\| \rangle_{(x,y) \sim \gamma}, \quad (3)$$

where Π is the set of all joint distributions $\gamma(x, y)$, with marginals P_T and P_G . The joint distribution can be understood as the “earth” that needs to be transported from x to y in order to transform P_T into P_G . The Wasserstein distance indicates the minimal cost of such

a transport. The distance of two non-intersecting distributions grows roughly linearly with their relative distance, leading to a stable gradient. Using the Kantorovich–Rubinstein duality [10], the Wasserstein distance can be reformulated as

$$W(P_T, P_G) = \max_{D \in \mathcal{D}} \langle D(x) \rangle_{x \sim P_T} - \langle D(\tilde{x}) \rangle_{\tilde{x} \sim P_G}. \quad (4)$$

The usual discriminator network is now replaced by a so-called critics network D . Its output is a 1-Lipschitz function which is trained to maximize $W(P_T, P_G)$. Since the definition of the EM distance depends on the maximization with respect to the critics network, the critics network is trained multiple times for each update of the generator. The Lipschitz condition can be met by clipping the weights of the critics if they exceed a maximum value. An improved version of the WGAN loosens the Lipschitz condition and replaces the weight clipping by the gradient penalty already mentioned for regular GANs [11, 12]. Wasserstein GANs are used in many particle physics applications [8, 13, 14].

An interesting GAN extension is cycle consistent GANs [15] which link two datasets, even though no direct correspondence of samples is given. Besides the standard one-directional mapping, the CycleGAN includes a second mapping in the inverse direction. Each mapping is trained with a corresponding discriminator, such that the mapped samples are indistinguishable from the respective target dataset. The second training objective is to achieve consistency, which means that the combination of both mappings results in the original input. If we have actual pairs of samples, we can directly use an invertible network which achieves the consistency automatically. We will explain this in more detail when discussing normalizing flows.

2.2. Variational autoencoder

An alternative approach to generating samples is variational autoencoders, consisting of an encoder network E and a decoder network D . In a simple autoencoder, the encoder maps the input to a latent representation, typically of reduced dimension, which the decoder maps back to the original sample. The training objective is to minimize

the reconstruction loss

$$L_{\text{AE}} = \|x - D(E(x))\|^2, \quad (5)$$

so that decoded samples become similar to true events. The decoder is trained to generate realistic samples from the latent space and could serve as a generator. Unfortunately, the standard autoencoder does not control the latent space, which means that realistic samples live in an arbitrary sub-space of the latent space. A variational autoencoder [16] organizes the latent space, for instance by enforcing Gaussian distributions. Instead of directly generating the latent representation, the encoder maps a data point to a multi-dimensional Gaussian characterized by vectors of mean values $\mu_j(x)$ and standard deviations $\sigma_j(x)$. In the limit of vanishing standard deviations, this gives us back the simple autoencoder. The VAE decoder is then applied to a sample drawn from this Gaussian distribution.

The corresponding extension of the loss function is motivated by variational inference. It can be derived minimizing the Kullback–Leibler (KL) divergence between the encoded distribution $q_x(z) = \mathcal{N}(\mu, \sigma)$ and the posterior $p(z|x)$. Under the assumption of a Gaussian prior, this loss simplifies to

$$\begin{aligned} L_{\text{VAE}} &= L_{\text{AE}} + \beta \cdot \text{KL}(q_x(z) | \mathcal{N}(0, 1)) \\ &= \|x - D(z)\|_{z \sim \mathcal{N}(\mu(x), \sigma(x))}^2 + \frac{\beta}{2} \sum_j 1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2. \end{aligned} \quad (6)$$

The free parameter β balances the relative importance of the reconstruction loss with respect to the enforcement of the prior. The authors of [17] choose small values of β to emphasize realistic samples. In this case, the encoded latent space no longer follows a Gaussian. Instead, they use a density information buffer to obtain a suitable prior distribution, from which they can sample new events.

Finally, one can combine concepts of GAN and VAE into adversarial autoencoders [18] or VAE-GANs [19]. The adversarial autoencoder replaces the KL term in the VAE loss with a discriminator

that distinguishes samples of the encoded distribution from a prior distribution. This allows us to choose arbitrary prior functions. The VAE-GAN replaces the reconstruction loss of the VAE by a discriminator that distinguishes reconstructed samples from the original data. This setup can generate sharper images when the MSE loss tends to have a blurring effect. While the optimal network architecture usually depends on the specific task and dataset, VAEs seem to be preferable when we require the additional control from the reduced latent space, while GANs tend to generate more realistic samples.

2.3. Normalizing flows and invertible networks

A third class of generative networks are normalizing flows [20–22], which use a bijective function f to transform a distribution of vector-valued random variables $x \in \mathbb{R}^D$ into a distribution of variables $y \in \mathbb{R}^D$ of the same dimension following a desired shape. The invertibility of each intermediate step makes the transformation traceable. This allows us to compute the probability density function (pdf) of the target variable y from the pdf of the input variable x . The access to the pdf of y is a prerequisite for the use of the network within Monte Carlo generators to improve integration and importance sampling [23–26].

We start with a random variable x following a probability distribution $p(x)$. The bijective function f in form of a network transforms the variable x to $y = f(x)$ and is parameterized with weights θ . The corresponding probability density function $q(y)$ is given by the substitution rule

$$q(y) = p(x) \left| \det \frac{\partial f}{\partial x} \right|^{-1}. \quad (7)$$

For practical purposes, the computation of the Jacobian determinant has to be efficient, while the transformation should be as expressive as possible. Initially proposed simple flows like planar and radial transformations [20] were soon replaced by more complex autoregressive flows like Real Non-Volume Preserving flows [27] (RealNVP). As proposed by the NICE framework [21] RealNVP rely on a triangular shape of the Jacobian to keep the determinant easily computable.

This is realized via so-called *coupling layers*, which split the input vector into two blocks $x = (x^A, x^B)$ using the partitions $\{A, B\}$ of the input dimension D . The output of the layer $y = (y^A, y^B)$, split into the same partitions, is given by

$$\begin{aligned} y_i^A &= x_i^A, \\ y_j^B &= C_j(x_j^B; m(x^A)), \end{aligned} \tag{8}$$

where the indices i, j run from 1 to $|A|, |B|$ respectively and the coupling transformation C is invertible. The Jacobian then takes a triangular form since C is separable, meaning the j th component of y_B depends only on the j th component of x^B

$$\frac{\partial f(x)}{\partial x^T} = \begin{pmatrix} \mathbb{1}_A & \mathbb{0} \\ \frac{\partial C_j(x_j^B; m(x^A))}{\partial x_i^A} & \frac{\partial C_j(x_j^B; m(x^A))}{\partial x_j^B} \end{pmatrix}. \tag{9}$$

The determinant is reduced to a simple product which can be computed within one forward pass. The exact form of C varies between implementations. Popular choices include affine and quadratic coupling layers. Since the Jacobian determinant of two consecutive mappings is simply given by the product of the individual Jacobians, one can combine multiple coupling layers to achieve a sufficient model capacity. The concept of autoregressive flows has since been further generalized in [28–30].

Once the normalizing flow is implemented, there is a multitude of different loss functions that can be used to train the network, for instance via the maximum likelihood approach [22]. Case studies to improve the SHERPA framework are trained by comparing the pdf of a sampled variable y with the true pdf at the same point obtained from the matrix element. They found a preference for the Pearson χ^2 divergence [26] and the exponential divergence [25] when training their networks.

The efficient calculation of the Jacobian is a necessary requirement to include normalizing flows into an integration routine, but the coupling layer offers the additional possibility to invert the full network. So far the described approach makes use of the invertibility,

but it never explicitly computes the inverse mapping of the network. While the computation is in principle possible for the general case described in Eq. (8), it can be computationally expensive, since the inversion of C can be arbitrarily complex. A suitable structure of C is given by invertible networks or INNs [31], a special type of normalizing flows for which the inversion of C is simple and the evaluation of the INN becomes very efficient in both directions. For instance, we can combine linear and exponential transformations to the invertible layer [27, 31]

$$\begin{aligned} y^B &= x^B \odot \exp(m_1(x^A)) + m_2(x^A) \\ \Leftrightarrow \quad x^B &= (y^B - m_2(x^A)) \odot \exp(-m_1(x^A)), \end{aligned} \quad (10)$$

where \odot indicates an element-wise multiplication.

We keep in mind that the sub-networks m_1 and m_2 , represented by a neural network, are evaluated only in the forward direction and remain unconstrained. Since the inversion does not require us to invert the sub-networks, we can condition them on an independent input without impact on the invertibility. This extension is called the conditional INN or cINN [32]. Its stability and its statistical properties make it particularly attractive to solve problems like unfolding detector effects and QCD jet radiation [33].

For such purposes, the cINN parameterizes again an invertible mapping between sampled variables y , which correspond to unfolded phase space points, and random numbers x . In addition, we now include conditional information c (corresponding to detector level information) via the subnets m_i . The cINN loss function is motivated by the simple argument that the final network parameters θ should maximize the (posterior) probability $p(\theta|y, c)$ or minimize

$$\begin{aligned} L &= -\langle \log p(\theta|y, c) \rangle_{y \sim P_y, c \sim P_c} \\ &= -\langle \log p(y|\theta, c) \rangle_{y \sim P_y, c \sim P_c} - \log p(\theta) + \text{const.} \\ &= -\left\langle \log p(f^{-1}(y, c)) + \log \left| \frac{\partial f^{-1}(y, c)}{\partial y} \right| \right\rangle_{y \sim P_y, c \sim P_c} \\ &\quad - \log p(\theta) + \text{const.} \end{aligned} \quad (11)$$

The second line uses Bayes' theorem and summarizes all terms independent of the minimization as constant. The third line simply applies the change of variables formula Eq. (7). When sampling over x the trained network finally yields correctly calibrated distributions over y under the condition c .

2.4. Amplification

An interesting question for neural networks in general, and generative networks in particular, is how much physics information the networks include in addition to the information in a statistically limited training sample. While a naive answer might be that all the physics a neural network can extract has to be encoded in the training data, the network setup adds information. For instance, it represents smooth functions up to a certain resolution. The question then becomes how much this very basic assumption accounts for in terms of events we can generate.

A simple, but instructive toy example is a one-dimensional camel back function [34], two Gaussians defined by two means, two widths, and a relative normalization, shown in the left panel of Fig. 2. The x -axis is divided into quantiles. For each of them we compute the

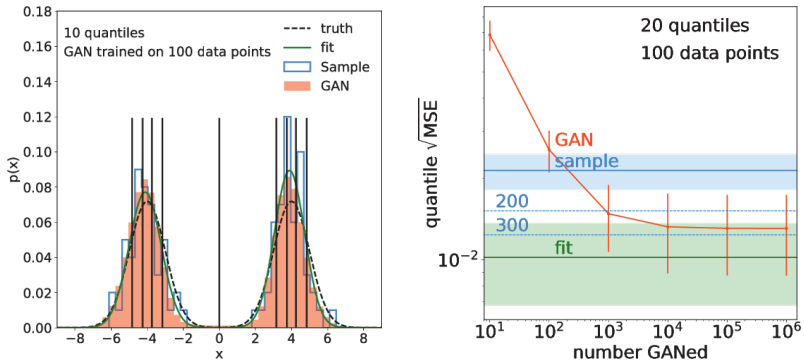


Fig. 2. Left: 1D camel back function, we show the true distribution (black), a histogram with 100 sample points (blue), a fit to the samples data (green), and a high-statistics GAN sample (orange). Right: quantile error for sampling (blue), 5-parameter fit (green), and GAN (orange), shown for 20 quantiles. Figures taken from [34].

statistical error in analogy to a χ^2 -measure and add those in quadrature. In the right panel, we show the combined error of all quantiles for the sample and for a 5-parameter fit benchmark. First, we see how the fit has a much smaller quantile error than the original 100-point sample. We can specify the additional information when we compare it to the number of sampled events we would need for the same quantile error. For the 20 quantiles in shown in the right panels of Fig. 2 the fit is worth around 500 events instead of the 100-event sample.

Clearly, a simple generative network will not an amplification factor of order five or even larger. Nevertheless, a GAN can be trained and then used to generate up to 10^6 events. We first see that generating more than 10,000 events does not change the quantile error and hence does not add more information. Second, we can read off the amplification factor and find that these 10,000 GANned events are worth almost 300 sampled events. In [34], the authors show that this kind of behavior extends to sparsely populated and high-dimensional phase space, and that the amplification factor increases with sparseness. The amplification factor of the GAN trails the amplification factor of the fit for the one-dimensional example. While a quantitative result on achievable amplification factors of generative networks in LHC simulations will depend on many aspects and parameters, this simple result indicates that using generative networks in LHC simulations can lead to an increase in precision.

3. Neural Networks in Event Generators

An obvious application of machine learning at the LHC is event generators. These generators are the simulation tools which put LHC physics into its unique position when it comes to understanding all aspects of the data and comparing it to first-principles theory predictions. Modules inside the generators describe the hard scattering, jet radiation, and even hadronization essentially from first principles. This means their input is a set of Lagrangians defined at a few distinct energy scales. Finally, the output from the event generators is fed into detector simulations, based on the detailed description of

the different sub-detectors. The numerical tool behind this generation chain is Monte Carlo simulations, which means that events are described by a long chain of random numbers which describe the individual steps independently from each other. As we will discuss in detail, modern machine learning offers many ways to improve such simulations. The practical question is where it can significantly speed up or increase the precision of the LHC simulation chain.

3.1. Phase space integration

One challenge in event generation at the LHC is the balance between global phase space coverage and the precise mapping of narrow local structures. The advantage of the established Monte Carlo methods is that they guarantee full phase space coverage, including regions where the matrix elements are very small. For a given algorithm, this global coverage has to be balanced with the local resolution, which means that we have to ensure that the event generator also resolves fine structures like phase space boundaries or intermediate resonance peaks. Algorithms like VEGAS [35] employ importance sampling, which means they adapt their grid of phase space points to the structures of the integrand and keep track of the Jacobian in terms of phase space weights. This method is nothing but a coordinate transformation of the phase space such that the Jacobian absorbs the main features of the integrand and the actual integration is now over a flat function. A prime example is the mapping of a Breit–Wigner propagator via

$$\int ds \frac{C}{(s - m^2)^2 + m^2\Gamma^2} = \frac{1}{m\Gamma} \int dz C \quad \text{with} \quad \tan z = \frac{s - m^2}{m\Gamma}. \quad (12)$$

The weak spot of VEGAS is that the adaptive phase space grid has a fixed rectangular form in the phase space dimensions. This can be improved by training a regression network to describe the mapping $s \rightarrow z$ such that the Jacobian of this variable transform absorbs the leading functional behavior of the integrand. In this case, the new integral will be over a largely constant function. Tools like

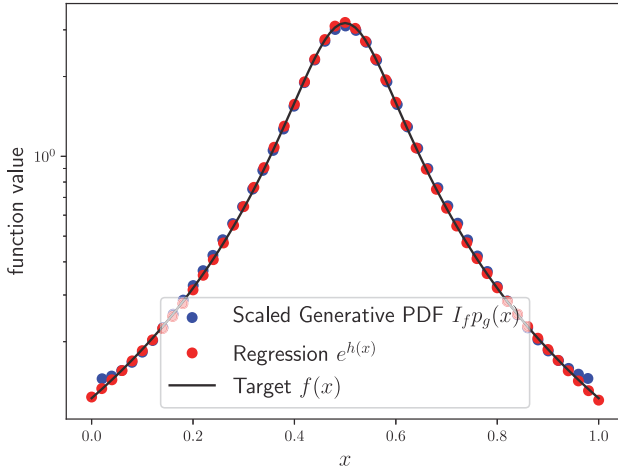


Fig. 3. Comparison of the target function value with the corresponding approximations from the regression and generative models. Figure taken from [37].

TENSORFLOW [36] provide this Jacobian essentially for free. References [37–39] show how neural network implementations can be used to integrate simple phase space structures extremely efficiently.

In [37], the author follows a slightly different approach and apply a generative network to evaluate the phase space integral. This GAN encodes the relation between a known, simple prior distribution and the integrand. In Fig. 3, we show how the regression network and the GAN map out the Breit–Wigner distribution of Eq. (12). For the example of a multi-dimensional camel function, the GAN integration outperforms not only VEGAS, but also a similar BDT implementation. A state-of-the-art version of a deep-learning integrator is i-flow [25, 40]. It uses a normalizing flow network and coupling layers to optimize the phase space mapping. The limitation of many of these studies is that they focus on phase space integration and not on phase space sampling or event generation. This means that for applications in LHC simulations we have to take the step from regression networks to generative networks discussed in Sec. 2. We will follow up this thought in Sec. 3.4.

3.2. Matrix elements

A main ingredient to event simulation is the form of the matrix element. We will discuss the features of matrix element estimation in more detail in Sec. 4 but mention some regression approaches already here. An early attempt of using machine learning on matrix elements targets the partonic process $gg \rightarrow ZZ$ [41]. Here the leading order is one loop, which means that the evaluation of the amplitude is significantly slower than the usual tree level calculations. At the same time, the simple $2 \rightarrow 2$ topology without intermediate resonance leaves us with a low-dimensional phase space and relatively flat distributions. While for the simple $2 \rightarrow 2$ scattering a BDT is sufficient to encode the matrix element, more complex processes as those discussed below require advanced machine learning tools. On the other hand, for instance NNLO calculations are limited by the calculation of loop-induced amplitudes, so this approach appears very promising.

A technically more sophisticated analysis targets the process

$$e^+e^- \rightarrow 3 \dots 5 \text{ jets} \quad (13)$$

to NLO [42]. For four or five jets in the final state, the precise calculation of the matrix element becomes computationally expensive. The question is how it can be encoded in a regression network, mapping the n -jet phase space onto the real value of the scattering amplitude. The key parameter is the pair-wise invariant mass of two partons, which diverges in the soft and collinear limits. The regression network features a MSE loss function and is implemented in KERAS [43] and TENSORFLOW [36] with the ADAM [44] optimizer.

The actual analysis focuses on a detailed study of the network uncertainties [45], especially in the critical, divergent-phase space regions. There the best regression networks achieve a precision of up to 1% in the value of the matrix element squared. As a systematic framework for error analyses, Bayesian networks also discussed in Chapter 18 have been applied to jet regression [46] and jet classification [47]. These analyses indicate that the framework can be applied in particle physics with its conservative frequentist approach.

A detailed comparison to the ensemble approach proposed in [42] could be a natural next step.

Divergent-phase space regions and their regularization with the help of subtraction terms are a known numerical challenge in LHC simulations. They can be treated with a subtraction GAN [48]. The task is to start with two different event samples and train a GAN such that its output follows a probability distribution given by the difference of the two training samples. In one dimension, this could be a base distribution P_B and a subtraction distribution P_S

$$P_B(x) = \frac{1}{x} + 0.1 \quad \text{and} \quad P_S(x) = \frac{1}{x}. \quad (14)$$

such that the GANned events follow the constant target distribution

$$P_{B-S} = 0.1. \quad (15)$$

In [48] this toy example is expanded to collinear subtraction with Catani–Seymour kernels, similar to the FKS subtraction used in [42]. The main difference between these two studies is that the former trains a generative network.

An alternative use for the subtraction GAN is studies of LHC signal processes. For instance, the kinematic distributions of Higgs decays to four fermions reflect the tensor structure of the Higgs coupling to gauge bosons. In traditional methods, we start from a combined sample of signal and background events and subtract the background events using some kind of naive or advanced side band analysis [49]. A subtraction GAN could be trained on the measured signal-plus-background sample and an appropriately prepared background sample and then produce signal events with all correlations. In Fig. 4, we show results for the simple example

$$\begin{aligned} B: & \quad pp \rightarrow \ell^+ \ell^- \\ S: & \quad pp \rightarrow \gamma \rightarrow \ell^+ \ell^-, \end{aligned} \quad (16)$$

such that $B - S$ gives the Z -induced contribution including the interference term. The GAN setup follows Ref. [7], discussed in Sec. 4.3. In passing, it also illustrates how GANs can surpass statistical limitations from the input samples, as we can see in the right panels

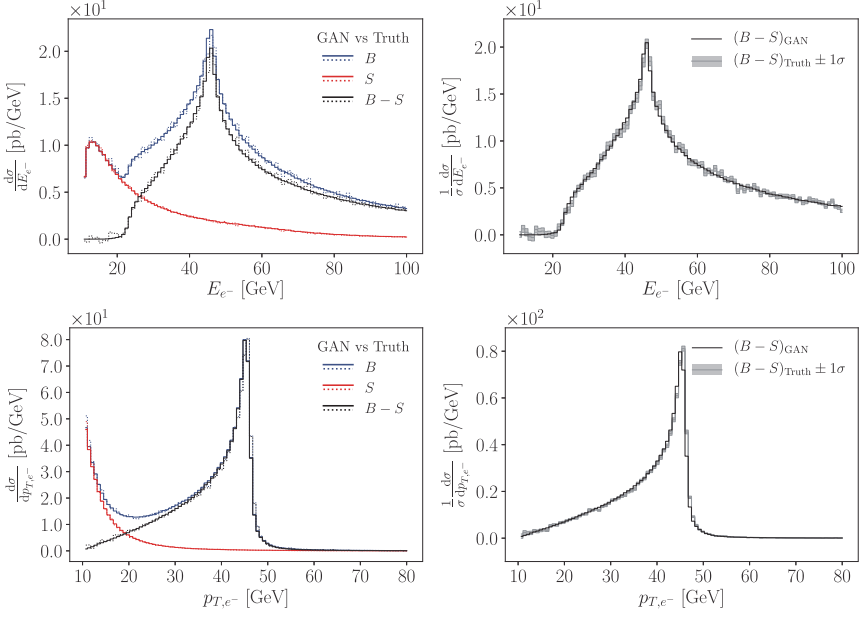


Fig. 4. Comparison of true and GANned $pp \rightarrow \ell^+ \ell^-$ events for the input samples and the GAN-subtracted sample. The right panels include the error envelope propagated from the input statistics. Figure taken from [48].

of Fig. 4. While the error envelope of the binned subtraction are given by the statistical uncertainty of the two original samples, the smaller variation of the GANned prediction benefits from the combined subtraction and interpolation.

3.3. Parton shower

The second step in an LHC event simulation is typically the treatment of jet radiation. It is also described by first-principles QCD, if we account for large soft and collinear logarithms [50]. The problems in describing it with a generative network are that it includes a very large number of particles in the final state, that it covers a wide range of energies, and that the self-similar structure of collinearly enhanced radiation needs to be accommodated. Eventually, there will be fully functional GAN showers for LHC analyses [51], but at this stage we

only discuss some early applications of neural networks in parton showers.

A standard way of representing jets in machine learning is jet images, two-dimensional pixelized images of the calorimeter output in the rapidity vs. azimuthal angle plane. Such images can be GANned using standard machine learning techniques, for instance loss functions which combine fake vs. truth discrimination with QCD vs. W -decay discrimination [52], also mentioned in Chapter 6. The training data for this jet image GAN are large-size PYTHIA8 [1] jets from QCD or from hadronic W -decays. They are required to be in the narrow range $p_T = 250 \dots 300$ GeV, to define a homogeneous sample. In addition, the jet images undergo basic pre-processing such that the hardest constituent is in the center and the second-hardest constituent is rotated to point down. The standard GAN setup is complemented by the additional class information about whether the jet comes from QCD or from W -decays. Because it operates on jet images, the network includes a set of convolutional layers, similar to the usual jet classification networks. It is implemented with KERAS [43] and TENSORFLOW [36] and uses the ADAM [44] optimizer.

A detailed study of the generated jets shows that they show promise in reproducing the relevant high-level observables like jet mass and subjettness sample-wise. An interesting way of testing if the GAN has learned the correct patterns is to train a classification network on truth or on GANned samples and then test this network on truth or GANned jets. It turns out that the GANned jets work well as a training sample, apparently too well, suggesting that the GAN has difficulties generating jets in the gray zone between typical QCD and typical W -decay jets. In Fig. 5, we show a detailed comparison of the 500 most signal-like and 500 most-background like jets out of 200k truth and GANned jets each. The two-dimensional histograms for the difference have a linear heat map. For these jets, the network reproduces the QCD and W -decay patterns faithfully, and some of the apparent differences are explained by bin migration.

Another early application of machine learning to parton showers [40] uses a regression network to apply an *a-posteriori* reweighting

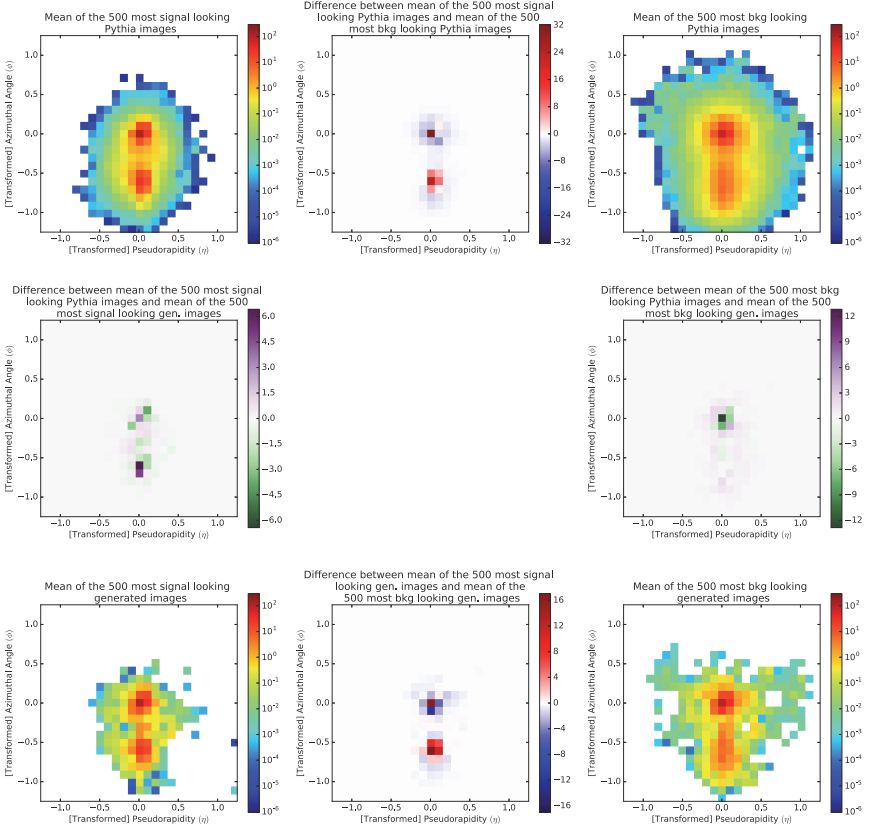


Fig. 5. Comparison between the 500 most signal- or W -looking (left) and most background- or QCD-looking (right) jet images, from the truth set (top) and the GANned set (bottom). Figure taken from [52].

to a parton shower. Examples are the reference value and the scale choice of $\alpha_s(\mu_R^2)$, which enters the parton shower in a non-trivial way. Varying these two parameters allows us to include theory uncertainties in an analysis of parton showers. The study finds that even a relatively simple network predicts the re-weighting factors for different observables with a precision of better than 2% with a promising gain in speed.

Our last example for using neural networks on parton showers generates Lund plane images using a GAN [8]. The starting points

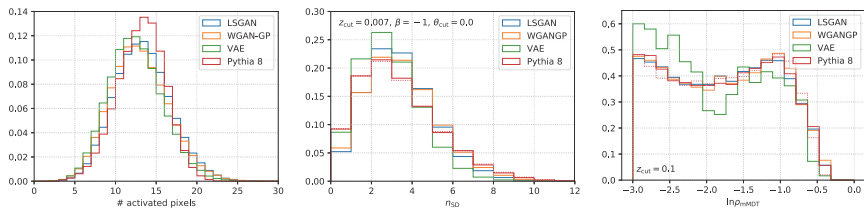


Fig. 6. Comparison of true and generated jets in terms of the number of activated pixels per image, the reconstructed soft-drop multiplicity, and the jet mass from the modified Mass Drop Tagger. Figure taken from [8].

are large jets with $p_T > 500$ GeV generated with PYTHIA8 [1] and then passed through the fast detector simulation DELPHES [53]. Each jet is then encoded through its clustering history in a sparsely populated two-dimensional image of the geometric, or R -separation and the relative transverse momentum. This two-dimensional representation, called Lund plane [54], is different from the usual jet images, which are defined as sparsely scattered pixels encoding the energy measured in calorimeter cells. The usual jet images encoding the calorimeter or even particle flow output define a starting point whenever we want to use machine learning on low-level observables. In contrast, Lund plane images represent the high-level output of a jet algorithm. The images are grouped into batches of 32 and used as training input to a least-square GAN, a gradient-penalty WGAN, and a VAE. The GANs employ a set of two-dimensional kernels. In Fig. 6, we compare the generated showers with the truth information in terms of different observables. While the two GANs lead to comparable results, the VAE performs visibly worse. Of the two GANs the LS version performs better when generating individual sparse Lund images rather than distributions over batches. At this stage, it is still too early to speculate what the optimal architecture for Lund plan images will be.

3.4. Sherpa and normalizing flows

The authors of the event generator SHERPA [3] have published two studies on how the phase space sampling could be improved using

deep learning. Both of them use normalizing flows with their invertible coupling layers. The authors of [24] start from the architecture of the i-flow integrator [25], implement it in SHERPA, and study the LHC process

$$pp \rightarrow W/Z + n \text{ jets.} \quad (17)$$

The neural network replaces the VEGAS-like importance sampling. Its task is to re-write an x -integration of a function $f(x)$ into a new variable x' such that the combination of the original integrand with the Jacobian, $w = f(x')/J$, is as close to a constant value over phase space as possible. All other parts of the SHERPA integration, including the multi-channel structure, remain the same. This implies that the sampling is still guaranteed to cover the full phase space. We recall that a standard generative network evaluates phase space following the training events, without any guaranteed coverage. Any improvement in constructing a phase space mapping by multi-dimensional interpolation should be visible in the unweighting efficiency of the phase space points. Motivated by the unweighting algorithms, the standard efficiency measure is the ratio of the average to the maximum event weights $\langle w \rangle / w_{\max}$, where the size of the denominator can be limited by evaluating it in batches.

In Table 1, we show the comparison of unweighting efficiencies with the standard SHERPA integrator and the i-flow network. It uses

Table 1. Unweighting efficiencies for the standard SHERPA integration and the normalizing flow network. Table slightly modified from [24].

Unweighting efficiency $\langle w \rangle / w_{\max}$		LO QCD					NLO QCD (RS)	
		$n = 0$	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 0$	$n = 1$
$W^+ + n \text{ jets}$	Sherpa	$3 \cdot 10^{-1}$	$4 \cdot 10^{-2}$	$8 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$8 \cdot 10^{-4}$	$1 \cdot 10^{-1}$	$5 \cdot 10^{-3}$
	NN + NF	$6 \cdot 10^{-1}$	$1 \cdot 10^{-1}$	$1 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$9 \cdot 10^{-4}$	$1 \cdot 10^{-1}$	$4 \cdot 10^{-3}$
	Gain	2.2	3.3	1.4	1.2	1.1	1.6	0.91
$W^- + n \text{ jets}$	Sherpa	$3 \cdot 10^{-1}$	$4 \cdot 10^{-2}$	$8 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$1 \cdot 10^{-3}$	$1 \cdot 10^{-1}$	$5 \cdot 10^{-3}$
	NN + NF	$7 \cdot 10^{-1}$	$2 \cdot 10^{-1}$	$1 \cdot 10^{-2}$	$2 \cdot 10^{-3}$	$8 \cdot 10^{-4}$	$2 \cdot 10^{-1}$	$4 \cdot 10^{-3}$
	Gain	2.4	3.3	1.4	1.1	0.82	1.5	0.91
$Z + n \text{ jets}$	Sherpa	$3 \cdot 10^{-1}$	$4 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	$5 \cdot 10^{-3}$		$1 \cdot 10^{-1}$	$5 \cdot 10^{-3}$
	NN + NF	$4 \cdot 10^{-1}$	$1 \cdot 10^{-1}$	$1 \cdot 10^{-2}$	$2 \cdot 10^{-3}$		$2 \cdot 10^{-3}$	$6 \cdot 10^{-3}$
	Gain	1.2	2.9	0.91	0.51		1.5	1.1

narrow jets with $p_T > 20$ GeV and $|\eta| < 6$, so a relatively large number of jets is expected in a typical LHC event, challenging the event generation. The gain in unweighting efficiency is clearly visible for the first two jets. Beyond this the flow network gains little, which contradicts the naive expectation based on an improved neural network interpretation for high-dimensional phase spaces. Instead, there seems to be a limiting factor to the performance of the flow network, which might have to do with the fact that all other parts of the generator, including the multi-channeling, are kept the same.

A second SHERPA study [26] also uses a normalizing flow network to replace the importance sampling module, but with a slightly different setup of the coupling layers. It studies the reference process

$$pp \rightarrow n \text{ gluons}, \quad (18)$$

also with small jets and $p_T > 30$ GeV. Here we know that the QCD (antenna) radiation pattern defines up to 120 Feynman diagram topologies or channels, which can be mapped onto three independent channels for $n = 4$. The analysis of the unweighting efficiencies confirms the bottom line of [24], namely that there is an improvement visible for $n = 3$, but not anymore for $n = 4$. This apparent breakdown is unexpected and needs more detailed studies.

In Fig. 7, we show a physics result from this study, namely the spectra of the three leading jets for $n = 4$. In the top panels, we see that the two importance sampling approaches, VEGAS and flow networks, both produce consistent results. Below, we see that also the MC uncertainty for the two approaches are consistent and remain below 2% as long as we stay away from the tails. Finally, in the bottom panes we show the mean weights $w = f/J$ introduced above. The perfect importance sampling would lead to a flat w -distribution over phase space, in this case unity everywhere. For the two leading jets both methods sample the tail too often, filling the histogram with many events of smaller weight. For the third jet, VEGAS starts to under-populate the tail while the flow network maintains the pattern from the leading two jets.

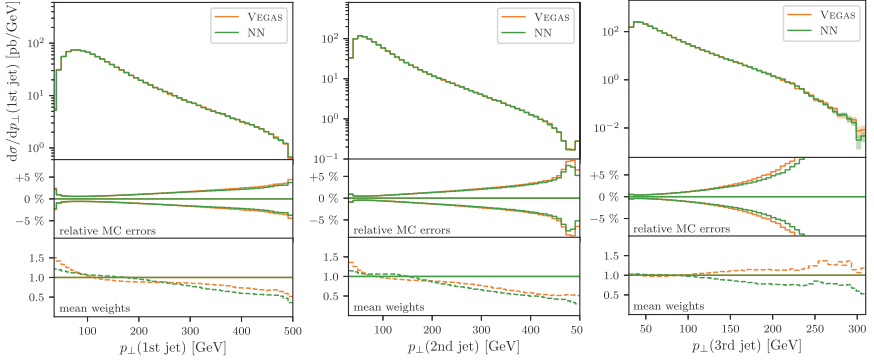


Fig. 7. Comparison of events from classic and flow network importance sampling in terms of p_T of the leading three jets (for up to four jets). The middle panes compare the Monte Carlo errors, the lower panes show the mean event weights per bin. Figure taken from [26].

An interesting aspect of this application of normalizing flows is that it does not use the invertible nature of the coupling layers. Instead, it benefits from the easy calculation of the derivative of the Jacobian. The integrator networks are similar to other generative networks in the sense that they map a random number input to phase space events. They do, however, produce weighted events, which by unfolding can be turned into unweighted events the same way they are produced by other generative networks. At this point, we only mention that event unweighting is a known weak spot of standard event generation, and using neural networks to generate unweighted events from a known phase space density might well be the most promising machine learning application within the standard simulation framework [55, 56].

4. GANs and VAEs as Event Generators

In simulating LHC events increased precision comes at a high price in computing. Leading order calculations are typically cheap, but can really only be considered order-of-magnitude estimates; NLO-QCD predictions have meaningful theory errors anywhere in the

20% to 50% range and are available through automated tools [3, 57]; precision analyses require NNLO or even N³LO in QCD and often require a wealth of numerical tricks to be used in LHC analyses, some of them involving machine learning, as discussed in Sec. 3.2. An alternative application of machine learning beyond improving generators is to train generative networks on any combination of simulated and actual events and then use their ability to learn and interpolate phase space structures to simulate large reference samples. We describe recent developments in this direction for three benchmark processes: the Drell–Yan process, multi-jet production, and top pair production at the LHC.

4.1. $Z \rightarrow \ell\ell$ production

Arguably, the best-studied standard candle at the LHC is the Drell–Yan process

$$pp \rightarrow \ell^+ \ell^- + \text{jets}, \quad (19)$$

where ℓ symbolizes visible leptons as well as invisible neutrinos, the latter being the leading background to dark matter searches.

In [58], the authors design a GAN to generate these events, described by the 4-vectors of two muons and up to five jets. In Sec. 3, we saw that for a sufficiently large number of jets this process is indeed a challenge and standard benchmark for Monte Carlo generators. The network is trained on PYTHIA8 [1] events including the fast detector simulation DELPHES [53] and a pile-up rate of 20 collisions on average. This simulation defines additional observable features which are evaluated for the network training, namely the number of primary vertices, the detector-induced missing transverse momentum vector, and the muon isolation.

The GAN employed for this paper includes a regression loss involving one process-specific feature, namely the position and the width of the Z -peak, in addition to the binary cross entropy

$$L = L_{\text{BCE}} + \lambda_m (m_Z - m_{\ell\ell})^2 + \lambda_\sigma (\sigma_Z - \sigma_{\ell\ell})^2, \quad (20)$$

with $\lambda_m = \lambda_\sigma = 10^{-4}$. The width $\sigma_Z = 7.7$ GeV is given by the detector simulation. The network is implemented in KERAS [43] with

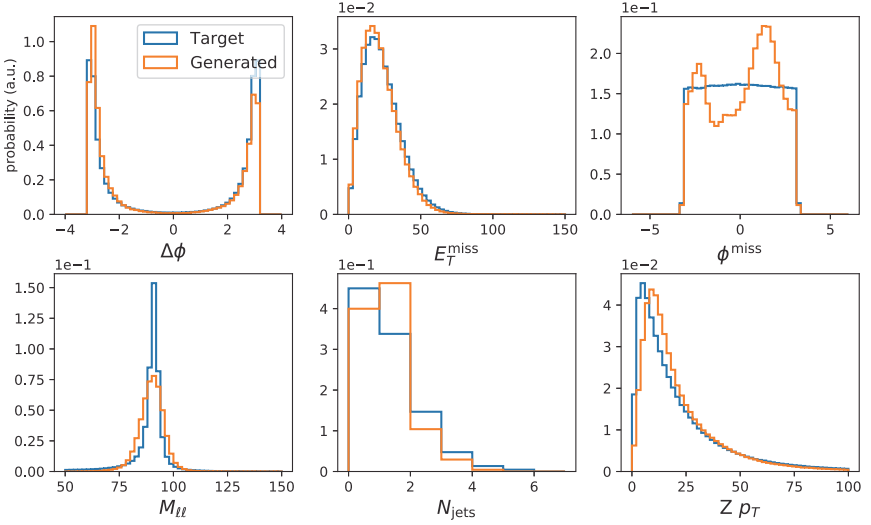


Fig. 8. Comparison of true and GANed $pp \rightarrow \ell^+ \ell^-$ events in terms of standard kinematic distributions. Figure taken from [58].

a TENSORFLOW [36] back-end, with a LeakyReLU activation. The comparison of the Z -mass and width goes beyond individual events and uses an event batch produced by the generator.

The quality of the GANed events can be tested with a list of kinematic observables, including the invariant mass of the two leptons and the number of jets with $p_T > 15$ GeV. The corresponding distributions are shown in Fig. 8. Removing the two Z -related terms from Eq. (20) has a negligible effect on the muon momenta and on their central invariant mass, but leads to an over-estimate of the detector-level Z -width by almost a factor of two. We will come back to on-shell mass peaks in Sec. 4.3. A problematic class of observables are the transverse jet momenta, because of the combination of the actual spectra and the peak from zero-padding events with fewer jets. Nevertheless, in the lower center panel of Fig. 8 we see that the number of jets above threshold is reproduced reasonably well at least up to three jets. Concerning the poorly learned azimuthal angle of the missing momentum direction, we speculate that the true distribution is essentially flat, but it is well known that generative networks often

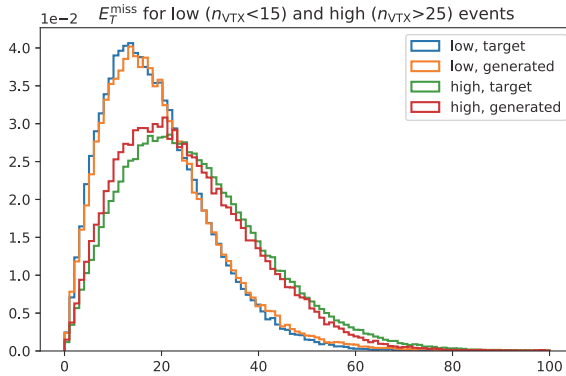


Fig. 9. Comparison of the E_T^{miss} distribution for true and GANned $pp \rightarrow \ell^+ \ell^-$ events in the low-pileup and high-pileup regime. Figure taken from [58].

fail to learn constant distributions over phase space [7]. The reason is that the combination of generator and discriminator updates will constantly force the two networks to move within a typical phase space distance and generate a noisy distribution.

An especially interesting aspect of [58] is the effect of pile-up, also studied in [59]. In Fig. 9, we show the missing transverse energy for two subsets of events, with low and high number of pile-up vertices. This application is an example for networks not enforcing energy-momentum conservation, which increases the dimensionality of phase space but allows for detector smearing. As we can see, the GAN reproduces the correlation between the number of pile-up vertices and the smearing of the detector-induced missing energy very well.

A similar physics process, but at an electron-positron collider

$$e^+ e^- \rightarrow Z \rightarrow \ell^+ \ell^- \quad (21)$$

is the starting point of [17]. The authors train on combined MG5AMCNLO samples [57] for $\ell = e, \mu$, where depending on the lepton flavor one set of 4-momenta is always set to zero. This setup increases the dimensionality of the final state from eight to 16. Because the simulation does not include detector effects, the $m_{\ell\ell}$ distribution now has a Breit-Wigner shape with the physics Z -width. This simulation also does not include any explicit information on the intermediate particle in the loss function.

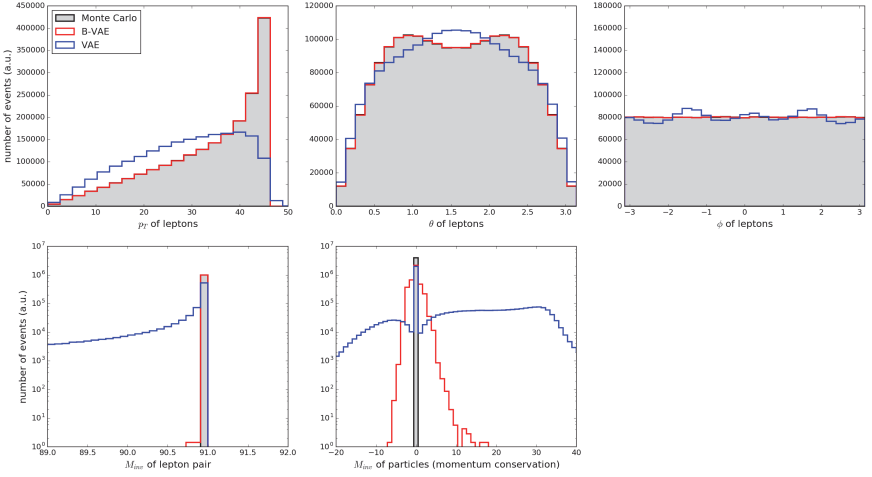


Fig. 10. Comparison of $e^+e^- \rightarrow \ell^+\ell^-$ events for the truth, a VAE with a standard Gaussian prior (blue) and the B-VAE (red). Figure taken from [17].

The generative network employed here is a modification of a VAE based on a combination of MSE and KL-divergence, as mentioned in Sec. 2. The so-called B-VAE developed for this purpose buffers density information in the latent space and is implemented with KERAS [43], TENSORFLOW [36] and CUDNN [60].

In Fig. 10, we show the corresponding kinematic distributions and confirm that unlike a naive VAE the B-VAE reproduces all of them. The last panel shows the invariant masses of the leptons, which should be zero and is now spread because the network learns the components of the external 4-vectors without the mass constraint. This observed smearing again reflects the above-mentioned problem of generative networks learning constants over phase space.

4.2. Multi-jets

Multi-jet production is the most frequent process at the LHC and affects a huge number of analyses. Depending on the kinematic cuts, the hard process includes at least two hard partons

$$pp \rightarrow q\bar{q}, gg, qg, \bar{q}g, \quad (22)$$

where these hard partons then generate at least two hard jets. Additional jets can be produced through hard scattering, initial state radiation, or final state radiation. Because of the logarithmic enhancement of collinear splittings and the relatively large strong coupling, most jet events at the LHC have many more than two jets [50]. Simple analyses study, for instance, the relative rate of n and $n+1$ jets, which can be predicted from QCD [50]. The challenges in simulating multi-jet events are, on the one hand, the variable number of jets in the final and, on the other hand, the required precision of a given analysis. The former comes from the fact that we cannot rely on counting powers of the strong coupling in perturbation theory, but have to re-sum large logarithms of jet radiation. The latter means that we have to combine fixed-order calculations with resummed calculation to high precision [50]. An alternative approach to simulating jet backgrounds could be generative networks describing this process based on data rather than theory simulations.

The authors of [61] train a GAN to simulated LHC events with at least two hard jets. The training data is simulated with MG5AMCNLO [57] and PYTHIA8 [1]. It relies on DELPHES [53] for fast detector simulation and FASTJET [62] for jet reconstruction. The large jet size of $R = 1.0$ ensures that there are not too many jets in the final state, for example from final state splittings. To enforce hard jets, all events are required to have a scalar sum of all transverse momenta $H_T > 500$ GeV.

The GAN is implemented in KERAS [43] and TENSORFLOW [36] with the ADAM [44] optimizer. All layers except for the last have a LeakyReLU activation function. In the input format the azimuthal angle of the leading jets is set to zero, exploiting a symmetry of the physical system. Another, symmetry is exploited through doubling the training data by reversing the rapidity.

In Fig. 11, we show a set of kinematic distributions for the training events and the generated events. The quoted χ^2 value quantifies the agreement between the respective true and GAN distributions. A typical feature of the multi-jet process is that most of the kinematic distributions are flat compared to processes with intermediate mass peaks. The only critical feature, already discussed in Sec. 3,

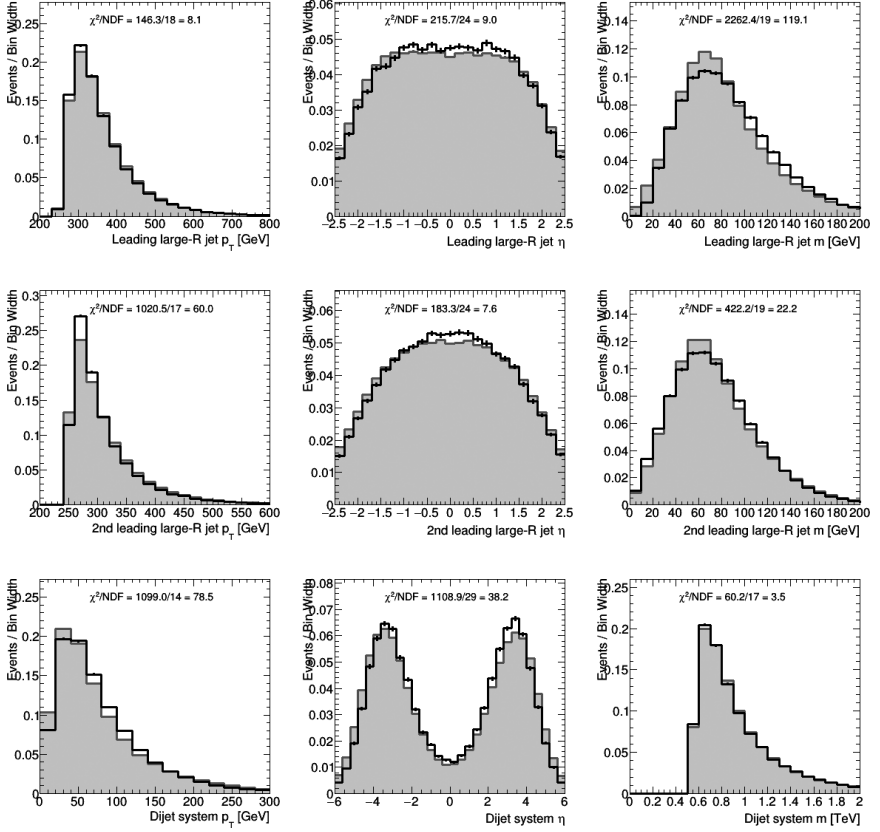


Fig. 11. Comparison of true (gray) and GANned (black) multi-jet events including detector effects. Figure taken from [61].

is the sharp phase space boundary for p_T^{\min} , in this case enforced through a cut on H_T and not fully aligned with the shown p_T . We know that a slight misalignment between a sharp boundary and the input parametrization helps the GAN to model the feature, because it softens the sharp edge. Nevertheless, there remains a slight deviation for instance around the p_T -threshold of the second-hardest jet. The last row of plots in Fig. 11 shows the kinematic recoil to the leading two jets. This recoil is generated by radiating a variable number of additional jets, so the results illustrate that the multi-jet GAN learns this variable number of jets.

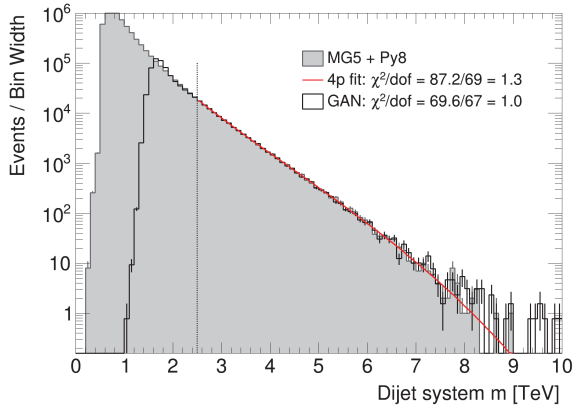


Fig. 12. Comparison of true (gray) and GANned (black) multi-jet events without detector effects. The red line shows a 4-parameter function fitted to the training data, including the high- m_{jj} tail. Figure taken from [61].

An interesting question lingering in all applications of generative networks is if networks can learn structures not only interpolating between phase space points, but extrapolating into poorly populated regions. For the dijet GAN [61] the authors train their model on a sub-set of the training data with $m_{jj} > 1.5$ GeV, this means they focus on the high-mass tail of the distribution and we can ignore issues in the low-mass range. In Fig. 12, we first show the training data, including a 4-parameter fit to the m_{jj} distribution as the baseline description. In addition, we show that the GANned events agree with the training data in the same m_{jj} distribution. The main difference appears for $m_{jj} \gtrsim 8.5$ TeV, where the number of training events becomes small, the fit function exhibits a sharp drop, and the GAN still provides a small number of events.

Finally, the B-VAE strategy of [17] illustrates for multi-jet production how an event sample can be generated from real data as opposed to simulated samples, in this case CMS data from a 7 TeV supersymmetry search [63]. In the original CMS paper, this jet sample has been shown to agree with a PYTHIA8 [1] multi-jet simulation, based on the hard di-jet process. The jet triggers effectively prefer leading jets with $p_T \gtrsim 100$ GeV and a sizeable di-jet mass. Missing transverse energy only appears through detector effects.

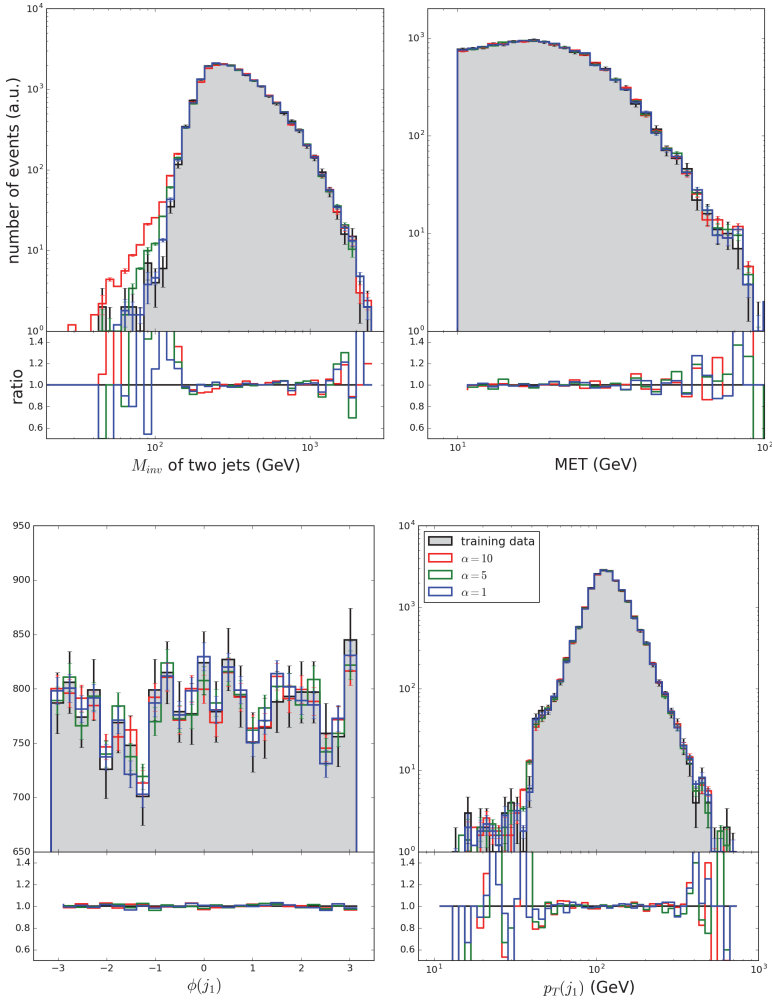


Fig. 13. Comparison between experimentally measured truth (gray) and B-VAE results (colored) for the CMS MultiJet primary dataset [63]. Figure taken from [17].

The employed B-VAE uses 4-momenta (E, p_T, η, ϕ) as input and operates on a 10-dimensional latent space for a variable number of standard jets. In Fig. 13, we show the agreement of the generated events with the original data. In contrast to typical simulated event samples, the CMS data does not have sharp phase space boundaries

or cliffs in a kinematic distribution. This allows the generative network to, for instance, describe the m_{jj} distribution over essentially the full range.

4.3. *Top pairs*

Top pair production at the LHC,

$$pp \rightarrow t\bar{t} \quad (23)$$

is an especially challenging process because it includes six particles in the final state, out of which we have to construct two intermediate W -propagators and two intermediate t -propagators.

In [17], the authors use their B-VAE to describe top pair production with one leptonic top decay. In that case the final state consists of exactly four jets and two leptons. The training data is produced with MG5AMCNLO [57] and supplemented with a fast detector simulation using DELPHES3 [53]. The 4-vectors are represented as (E, p_T, η, ϕ) , defining a 26-dimensional phase space including the two parton-momentum fractions x . Hyper-parameters which need to be optimized for the B-VAE include the B -parameter weighting the MSE and KL-divergence in the loss function and the dimensionality of the latent space. It is interesting to note that the best-performing models for a set of one- and two-dimensional kinematic distributions in [17] have an approximately 20-dimensional latent space.

In Fig. 14, we show some of the kinematic distributions, describing the final state particle in the upper row and correlating the final state particles in the lower row. In general, the B-VAE learns the features of the production process. The challenge in the transverse momentum distribution, as compared to the rapidity, is the sharp drop-off for small $p_{T,j}$. Such sharp features or even phase space boundaries are a known and obvious challenge for any generative network [7, 64]. The reason is that the end of such a distribution is described by a very small number of events, so the network will be limited by the training statistics. Good examples for smooth distributions are $\eta_{J,1}$, $\Delta\phi(\ell, \text{MET})$, or $\Delta R(j_1, j_2)$ where the precision of the B-VAE is shown to be around 10% at least.

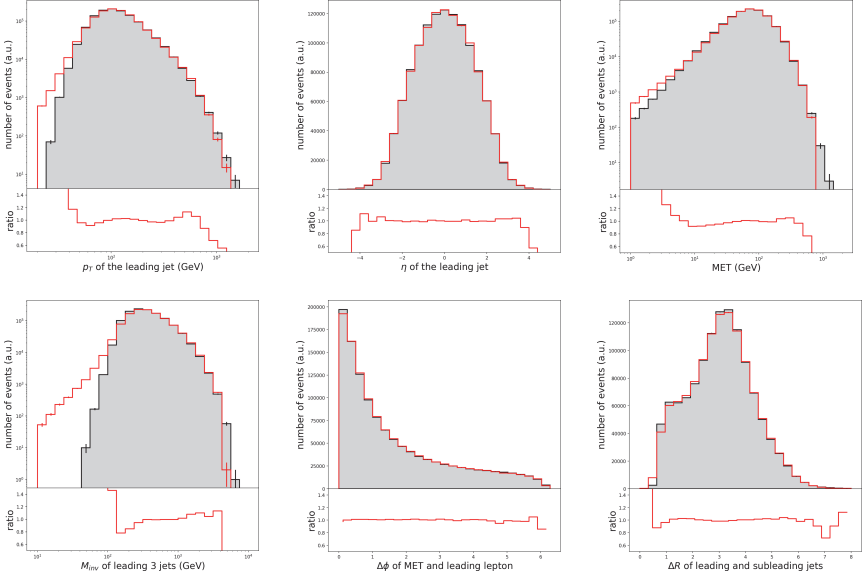


Fig. 14. Comparison of true (gray) and VAE (red) events for $t\bar{t}$ production. We show a subset of distributions from [17].

The $t\bar{t}$ study in [7] focuses on an open question from the results shown in [17] and an obvious problem found in [58], namely intermediate on-shell resonances. These narrow phase space features are also a known problem for standard matrix element integrators, which typically employ dedicated coordinate transformations or (multi-channel) phase space mappings. In this case, the training data are top pair events simulated with MG5AMCNLO [57], now decaying into an all-hadronic final state. As a simplification, events with additional jets are not considered. A detector simulation would lead to broader intermediate mass peaks, so it is omitted in reference to the main challenge of the analysis.

The input to the network are the six 4-vectors (E, p_x, p_y, p_z) , but with an explicit on-shell condition for each final state particle. They are fed into a GAN with a gradient penalty, implemented in KERAS [43] and TENSORFLOW [36]. The gradient penalty stabilizes the training to a level comparable with a Wasserstein GAN.

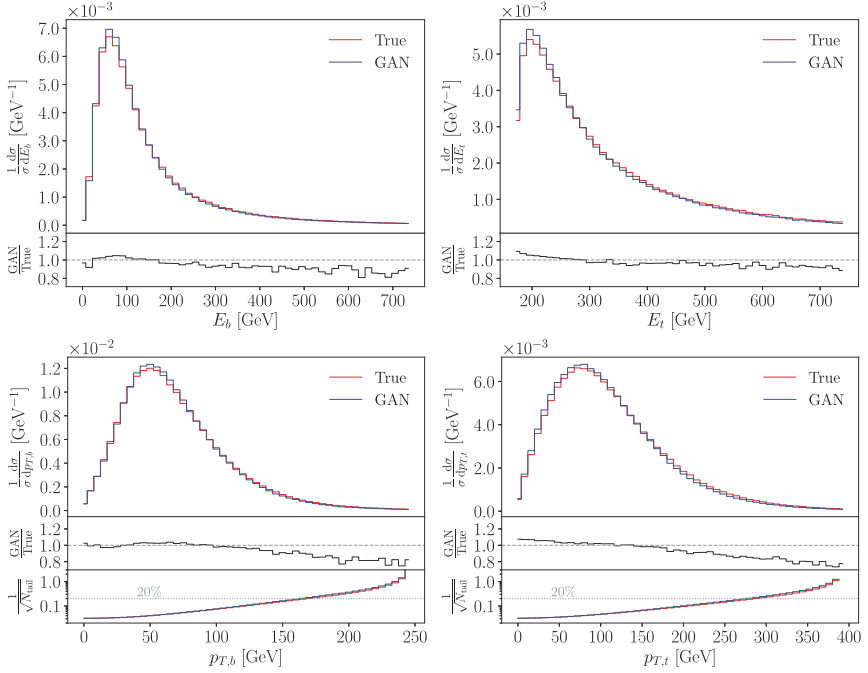


Fig. 15. Comparison of true and GANned events. The additional panels give the bin-wise ratio. The third panels show the statistic uncertainty on the number of training events in the tails. Figure taken from [7].

Some kinematic distributions are shown in Fig. 15, again indicating an agreement with the training data at the 10% level.

Coming back to the main challenge, invariant masses, like many other narrow phase space features, can be cast into well-defined one-dimensional distributions. In the loss function such a distribution can, for instance, be enforced through a maximum mean discrepancy (MMD) [65], a kernel-based method to compare two samples drawn from different distributions. Using one batch of true data points following a distribution P_T and one batch of generated data points following P_G , it computes a distance between the distributions

$$\text{MMD}^2 = \langle k(x, x') \rangle_{x, x' \sim P_T} + \langle k(y, y') \rangle_{y, y' \sim P_G} - 2 \langle k(x, y) \rangle_{x \sim P_T, y \sim P_G}, \quad (24)$$

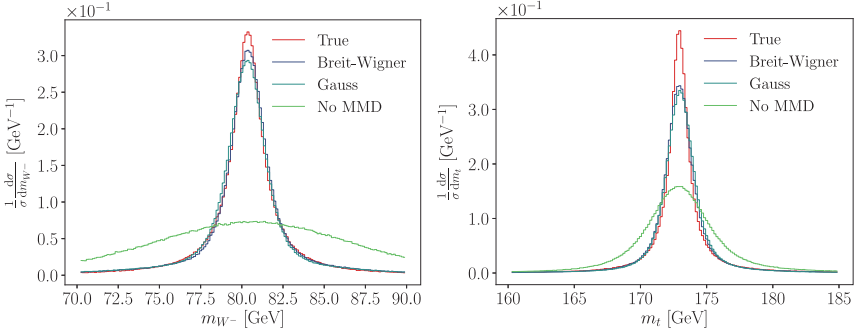


Fig. 16. Comparison of different kernel functions for the W -boson and top mass peaks in the top-pair GAN. Figure taken from [7].

where $k(x, y)$ can be for instance Gaussian or Breit–Wigner kernels. In both cases, the kernel width is a hyperparameter of the network. We show the effect of the different kernels in Fig. 16.

Finally, to show that astronomy is not the only field producing nice-looking pictures, we also compare a two-dimensional correlation between the true data and the GAN output in Fig. 17. The correlation between the two transverse momenta includes a Jacobian peak as well as a sharp phase space boundary. The slice in the lower-right panel indicates that the GAN learns the Jacobian peak as well as the sharp boundary with high precision.

5. Inverting the Simulation Chain

While the LHC simulation chain discussed in Sec. 3 is statistically invertible, it is only ever applied in one direction: we define a physics hypothesis for instance at the hard matrix element level, derive predictions for a dataset, and compare with measured data. This procedure turns around our actual physics question, which for instance asks how a kinematic distribution, assuming a hard process, looks for a measured dataset. For the interaction between theory and experiment it would therefore be extremely useful, if we could move up and down the simulation chain and compare measurement and theory at any level of data processing.

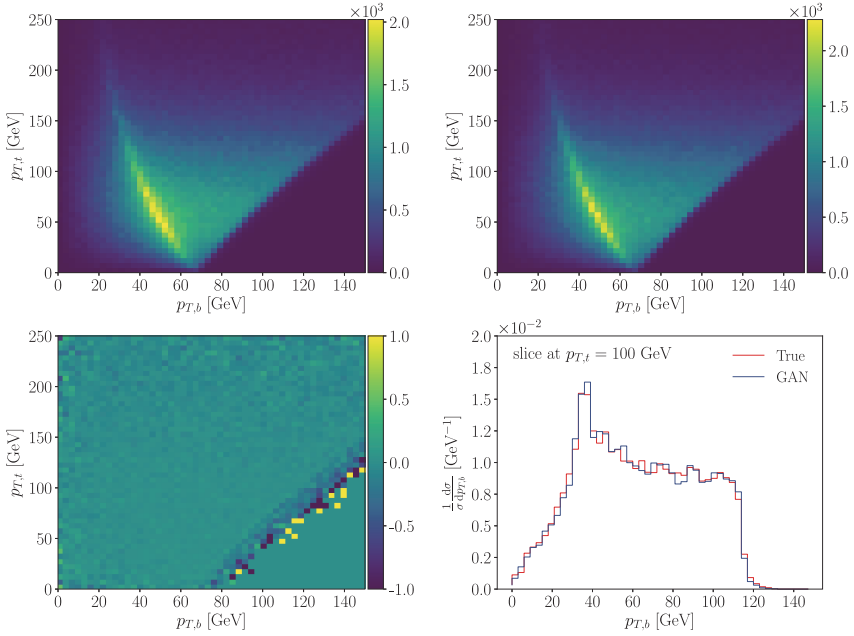


Fig. 17. Correlation between $p_{T,t}$ and $p_{T,b}$ for truth (upper left), GAN (upper right), and their relative difference (lower left). We also show $p_{T,b}$ sliced at $p_{T,t} = 100 \pm 1$ GeV. Figure taken from [7].

A simple case would be inverting detector effects, starting from detector-level events and showing parton-level kinematic features. This special case is called unfolding detector effects, and it is an established procedure for one or two-phase space dimensions. Similarly, analyses based on estimating parton-level matrix elements are known as using the matrix element method [66–68]. The hope is that inverting the LHC simulation chain with machine learning will open new ways to analyze LHC data and compare it to theory predictions without always implementing them into event generators.

5.1. Parton shower from CycleGANs

When we model an, in principle, invertible simulation like event generation with a neural network, we actually have to decide in which direction we want to apply the network. An intuitive way out is to

define a network which maps the incoming dataset to the outgoing dataset and back. An example is given in [8], where a CycleGAN turns QCD jets and W -decay jets into each other. Alternatively, the same CycleGAN can apply and invert detector effects on a set of jets.

Specifically, the training data are QCD jets and W -decay jets from PYTHIA8 [1], which are passed through DELPHES [53]. Each jet is represented by a Lund plane image, introduced in Sec. 3.3. The mapping of a sample of QCD jets onto a sample of W -jets (and vice versa) could help in providing a realistic and large set of fat jets at low simulation cost, similar to the generative networks discussed in Sec. 4. The difference to the other generator models is that it works on a sample of QCD jets, not from scratch. This relieves the network from having to learn the basic structure of a jet and should speed up the generation. On the other hand, a pre-defined structure always bears the danger of introducing a bias into the network.

The, arguably, more interesting application is the unfolding of non-perturbative QCD effects and detector effects from a set of observed jets. We show an illustration of this task in the upper panels of Fig. 18. Because Lund images are defined as superpositions of jet batches we sample individual jets from the images at parton level and at detector level. We show individual jets generated from the Lund images in the lower panels of Fig. 18.

A similar approach to unfolding detector effects starting from a good first estimate and then iterating improvement steps has been developed for full LHC events. This OMNIFOLD [69] approach starts with pairs of simulated events at parton level and at detector level, constructs a mapping between simulated and measured detector-level events, and applies this mapping to the parton-level simulations. The output are parton-level events corresponding to measured events, and the procedure is improved through an iteration. This iteration removes a possible bias from the original paired events.

5.2. Detector unfolding with FCGANs

Using generative networks to directly unfold detector effects from LHC events was first proposed in [70]. A first, properly generative

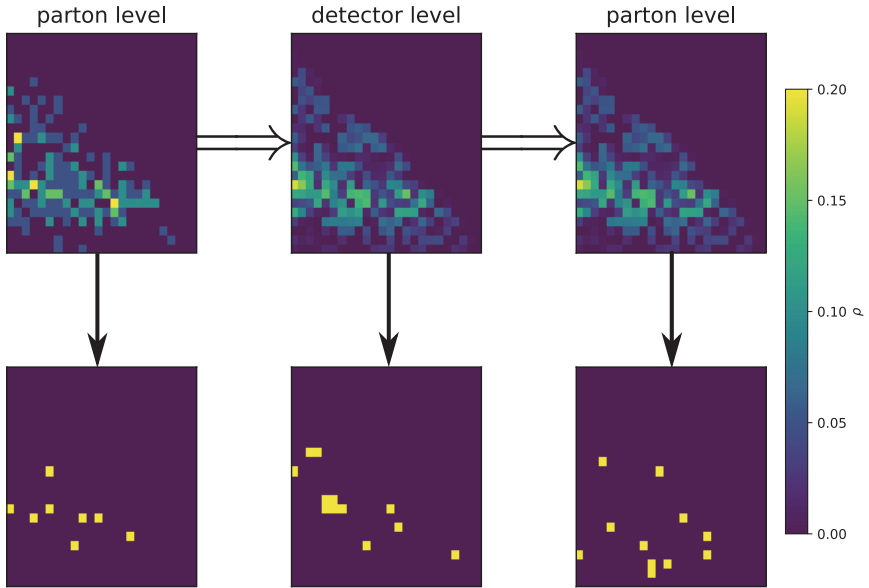


Fig. 18. Top: jet translation from parton-level to detector-level and back. Bottom: corresponding sampled event. Figure taken from [8].

approach was then established for the process [71]

$$pp \rightarrow WZ \rightarrow (q\bar{q}') (\ell^+ \ell^-), \quad (25)$$

trained on Standard Model events generated with MG5AMCNLO [57] and PYTHIA8 [1]. These parton-level events are then fed through DELPHES [53] and FASTJET [62] for the jet reconstruction. The analysis does not allow for additional jet radiation, postponing this issue to the analysis discussed in Sec. 5.3. The task is to train a generative network on a sample of paired parton-level and detector-level events such that the network generates statistically correct parton-level events from a detector-level event. The detector-level event is represented by 4-vectors of high-level analysis objects, like leptons and jets. This detector unfolding has two shortcomings: first, it is only defined statistically in the sense that it does not produce a probability distribution in parton-level phase space for a given detector-level event. Second, it always assumes an underlying physics hypothesis,

in our case the Standard Model describing the hard scattering in the training data.

As long as the network is applied to detector-level events which are essentially identical to the training data, the naive GAN approach following [70] will work fine. Its architecture follows the event generation GAN in Sec. 4.3. A problem appears if the test and training datasets are not quite identical. Because the unfolding GAN does not have a notion of similarity in terms of event kinematics, for instance in terms of a latent space metric, it will fail [71]. A way out is to replace the GAN with a fully conditional FCGAN, trained to reproduce a parton-level event only from random noise under the condition of the matching detector-level event with all its physics information. We show the results from this FCGAN in Fig. 19, applied to test

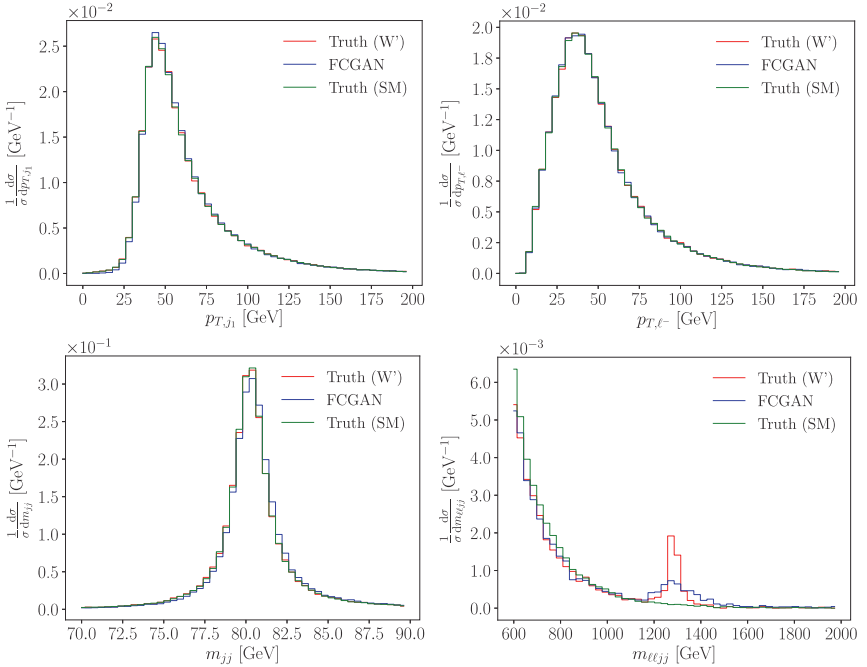


Fig. 19. Comparison of parton-level truth and FCGANned distributions for the process $pp \rightarrow WZ \rightarrow q\bar{q}\ell\ell$. The network is trained on the Standard Model and used to unfold events with an injection of 10% W' events with $m_{W'} = 1.3$ TeV. Figure taken from [71].

data including an irreducible resonance contribution

$$pp \rightarrow W' \rightarrow WZ \rightarrow (q\bar{q}') (\ell^+ \ell^-). \quad (26)$$

While the network does not reproduce the W' -width correctly, it clearly shows the mass peak which did not exist in the training data. This serves as an indication that it is possible to unfold detector-level events with a controllable model dependence and hence to apply this technique to new physics searches.

5.3. *Hard process from cINNs*

An alternative approach to inverting detector effects is based on invertible networks, using the setup described in Sec. 2. The INN can be trained in the well-defined DELPHES [53] direction, mapping parton to detector-level events, and evaluated in the inverse direction to unfold the detector-level distribution [33]. If parton-level and detector-level events live in phase spaces with different dimensions, the smaller representation is extended with noise parameters. Since our task is to construct a non-deterministic mapping, we can try to include more random numbers into the network input and output. Finally, we will use a generative network that includes the foundation of statistical sampling already in the loss function.

The system is benchmarked on the same detector unfolding problem as in Sec. 5.2 and focus on the statistical interpretation. In the left panel of Fig. 20 we show the distribution in the unfolded parton-level phase space, specifically p_{T,q_1} for 3200 independent unfoldings of the same pair of parton-level and detector-level events. First, the FCGAN approach does not allow for a statistical interpretation of the results. While the FCGANned events reproduce the correct kinematic distributions at the parton level, it is not possible to invert a single detector-level event and obtain something like a posterior probability distribution. After padding the standard INN input vectors with a sufficiently large number of random numbers, the so-defined noise-extended eINN does produce a reasonably distribution in parton-level phase space. We can test the width of this distribution through a calibration test: for the right panel of Fig. 20 1500 pairs of parton-level and detector-level events are unfolded 60 times each.

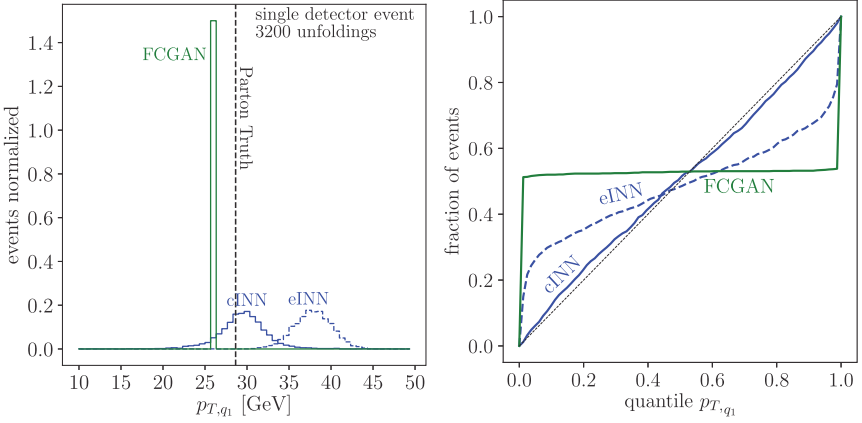


Fig. 20. Left: illustration of the statistical interpretation of unfolded events for one event. Right: calibration curves for $p_{T,q1}$ extracted from a conditional GAN, a noise-extended eINN, and a conditional cINN. Figure taken from [33].

For each of them we can look at the position of the parton-level truth in the unfolded distribution, expecting 10% of the 1500 event to lie within the 10% quantile from the left, 20% in the 20% quantile, etc. In the left panel of Fig. 20 we see, however, that the eINN distribution is too narrow to cover the truth. In the right panel, we confirm this shortcoming in that the eINN output need re-calibration.

We already know that for a statistically sound approach we can try a conditional (invertible) network. As for the FCGAN the direct mapping between parton level and detector level is replaced by a conditioned mapping between parton-level observables and a random variable of the same dimension. Also in Fig. 20 we show the results from this cINN and find that it provides posterior probability distributions with an almost perfect calibration. Modulo an unavoidable model dependence, these studies show that it is possible to compute probability distributions over parton-level phase space for single detector-level events.

An additional benefit of the cINN is that the detector-level input can be of arbitrary dimension. Technically, this makes it possible to unfold events with any number of additional jets [33],

$$pp \rightarrow WZ + \text{jets} \rightarrow (q\bar{q}') (\ell^+ \ell^-) + \text{jets}. \quad (27)$$

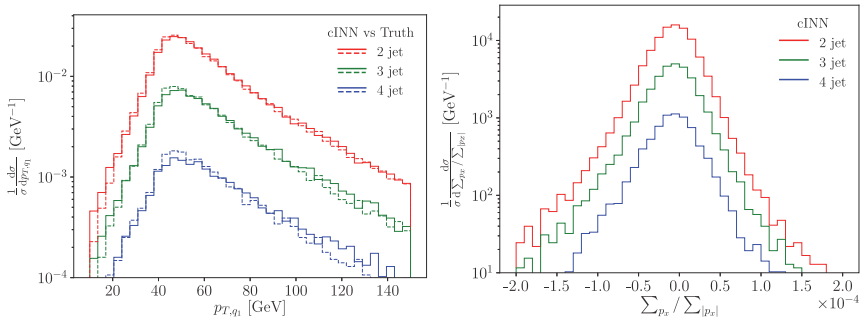


Fig. 21. Comparison of parton-level truth and cINNed distributions for the process $pp \rightarrow (q\bar{q}') (\ell^+\ell^-) + \text{jets}$. The network is trained on detector-level events with two to four jets. The parton-level events are stacked by number of jets at detector level. Figure taken from [33].

The number of jets in the hard process has to be defined as part of the unfolding model. This flexibility is crucial to include perturbative QCD corrections in the parton-level theory prediction. The stacked $p_{T,q1}$ distribution in Fig. 21 shows how the network unfolds 2-jet, 3-jet, and 4-jet events with similar precision. In the right panel, we see that at all unfolded events respect transverse momentum conservation at the level of the hard $2 \rightarrow 2$ process. Going back to the topic of the review, this last example shows that we cannot just generate events using neural networks, but that we can also invert the generation chain for the LHC. This is a very significant advantage over the usual simulation methods as it allows for completely new ways to compare theory predictions and measured data for future LHC runs.

6. Outlook

We have discussed the application of generative neural networks to event generation for example at the LHC. In the standard approach, this is done with Monte Carlo simulations which use Lagrangians as inputs and provide simulated LHC events based on first principles. This approach guarantees full phase space coverage, but it is becoming speed-limited and cannot be inverted in practice. This implies that analyses can only be done at the end of the simulation chain.

We have discussed many ideas to improve and complement this simulation chain using neural networks. In Sec. 3, we have shown how neural networks can be used as modules in contemporary event generators, from phase space simulation to matrix elements and parton showers. Next, we discussed in Sec. 4 how this event generation chain might be replaced by generative networks. We note that this does not imply that neural networks will replace first-principle generators, because only first-principle generators allow us to compare LHC data to complete theory predictions. Instead, event generation networks could be used to increase the number of simulated events or to cover statistical weaknesses of standard simulators for instance in the bulk of high-precision simulations.

Finally, we have discussed how neural networks can invert the simulation chain for the LHC. Such an inversion is at the heart of approaches like the matrix element method. Moreover, a systematic unfolding would enable analyses at any level of the LHC simulation chain and give the experiments access to many more precision predictions. These applications of machine learning to LHC simulations are still at the very beginning, and many conceptual problems are unsolved. For instance, it is not clear how many events a trained network can generate before it is limited by the limited size of the training data, and we do not know how to assign error bars to event samples generated by neural networks. On the other hand, the existing studies clearly indicate the potential of neural networks as part of simulation tools, and there is no doubt that LHC simulations and simulation-based analyses during the upcoming runs will significantly benefit from generative networks.

Acknowledgments

We would like to thank all of our collaborators and discussion partners on generative networks, including Lynton Ardizzone, Marco Bellagente, Sascha Diefenbacher, Gregor Kasieczka, Ulli Köthe, Ben Nachman, Armand Rousselot, and especially Ramon Winterhalder. We are also grateful to David Rousseau, Paolo Calafiura, and Kazuhiro Terao for giving us this opportunity and to Paolo Calafiura

for great input on the motivation section. Our research is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 — TRR 257 *Particle Physics Phenomenology after the Higgs Discovery*.

References

- [1] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen and P. Z. Skands, An introduction to PYTHIA 8.2, *Comput. Phys. Commun.* **191** (2015) 159; arXiv:1410.3012 [hep-ph].
- [2] J. Bellm *et al.*, Herwig 7.2 release note, *Eur. Phys. J. C* **80**(5) (2020) 452; arXiv:1912.06509 [hep-ph].
- [3] Sherpa, E. Bothmann *et al.*, Event generation with Sherpa 2.2, *SciPost Phys.* **7**(3) (2019) 034; arXiv:1905.09127 [hep-ph].
- [4] J. Alwall *et al.*, The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations, *J. High Energy Phys.* **07** (2014) 079; arXiv:1405.0301 [hep-ph].
- [5] HSF Physics Event Generator WG, S. Amoroso *et al.*, Challenges in Monte Carlo event generator software for high-luminosity LHC, preprint (2020); arXiv:2004.13687 [hep-ph].
- [6] P. Calafiura, J. Catmore, D. Costanzo and A. Di Girolamo, ATLAS HL-LHC computing conceptual design report, Technical Report CERN-LHCC-2020-015. LHCC-G-178, CERN, Geneva, September (2020); <http://cds.cern.ch/record/2729668>.
- [7] A. Butter, T. Plehn and R. Winterhalder, How to GAN LHC events, *SciPost Phys.* **7**(6) (2019) 075; arXiv:1907.03764 [hep-ph].
- [8] S. Carrazza and F. A. Dreyer, Lund jet images from generative and cycle-consistent adversarial networks, *Eur. Phys. J. C* **79**(11) (2019) 979; arXiv:1909.01359 [hep-ph].
- [9] M. Arjovsky, S. Chintala and L. Bottou, Wasserstein GAN, preprint (2017); arXiv:1701.07875 [stat.ML].
- [10] C. Villani, *Optimal Transport: Old and New*, Grundlehren der mathematischen Wissenschaften (Springer, Berlin 2008).
- [11] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin and A. Courville, Improved training of wasserstein GANs (2017); arXiv:1704.00028 [cs.LG].
- [12] L. Mescheder, A. Geiger and S. Nowozin, Which training methods for GANs do actually converge? arXiv:1801.04406 [cs.LG].
- [13] M. Erdmann, L. Geiger, J. Glombitza and D. Schmidt, Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks, *Comput. Softw. Big Sci.* **2**(1) (2018) 4; arXiv:1802.03325 [astro-ph.IM].

- [14] M. Erdmann, J. Glombitza and T. Quast, Precise simulation of electromagnetic calorimeter showers using a Wasserstein generative adversarial network, *Comput. Softw. Big Sci.* **3**(1) (2019) 4; arXiv:1807.01954 [physics.ins-det].
- [15] J.-Y. Zhu, T. Park, P. Isola and A. A. Efros, Unpaired image-to-image translation using cycle-consistent adversarial networks, preprint (2017); arXiv:1703.10593 [cs.CV].
- [16] D. P. Kingma and M. Welling, Auto-encoding variational bayes, preprint (2013); arXiv:1312.6114 [stat.ML].
- [17] S. Otten, S. Caron, W. de Swart, M. van Beekveld, L. Hendriks, C. van Leeuwen, D. Podareanu, R. Ruiz de Austri and R. Verheyen, Event generation and statistical sampling for physics with deep generative models and a density information buffer, preprint (2019); arXiv:1901.00875 [hep-ph].
- [18] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow and B. Frey, Adversarial autoencoders, preprint (2015); arXiv:1511.05644 [cs.LG].
- [19] A. B. L. Larsen, S. K. Sønderby, H. Larochelle and O. Winther, Autoencoding beyond pixels using a learned similarity metric, preprint (2015); arXiv:1512.09300 [cs.LG].
- [20] D. J. Rezende and S. Mohamed, Variational inference with normalizing flows, arXiv:1505.05770 [stat.ML].
- [21] L. Dinh, D. Krueger and Y. Bengio, Nice: Non-linear independent components estimation, preprint (2014); arXiv:1410.8516 [cs.LG].
- [22] I. Kobyzev, S. Prince and M. A. Brubaker, Normalizing flows: An introduction and review of current methods, preprint (2019); arXiv:1908.09257 [stat.ML].
- [23] T. Müller, B. McWilliams, F. Rousselle, M. Gross and J. Novák, Neural importance sampling, preprint (2018); arXiv:1808.03856 [cs.LG].
- [24] C. Gao, S. Hoeche, J. Isaacson C. Krause and H. Schulz, Event generation with normalizing flows, preprint (2020); arXiv:2001.10028 [hep-ph].
- [25] C. Gao, J. Isaacson and C. Krause, i-flow: High-dimensional integration and sampling with normalizing flows, preprint (2020); arXiv:2001.05486 [physics.comp-ph].
- [26] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale and S. Schumann, Exploring phase space with neural importance sampling, preprint (2020); arXiv:2001.05478 [hep-ph].
- [27] L. Dinh, J. Sohl-Dickstein and S. Bengio, Density estimation using real NVP, preprint (2016); arXiv:1605.08803 [cs.LG].
- [28] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever and M. Welling, Improving variational inference with inverse autoregressive flow, preprint (2016); arXiv:1606.04934 [cs.LG].
- [29] G. Papamakarios, T. Pavlakou and I. Murray, Masked autoregressive flow for density estimation, preprint (2017); arXiv:1705.07057 [stat.ML].
- [30] C.-W. Huang, D. Krueger, A. Lacoste and A. Courville, Neural autoregressive flows, preprint (2018); arXiv:1804.00779 [cs.LG].

- [31] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother and U. Köthe, Analyzing inverse problems with invertible neural networks (2018); arXiv:1808.04730 [cs.LG].
- [32] L. Ardizzone, C. Lüth, J. Kruse, C. Rother and U. Köthe, Guided image generation with conditional invertible neural networks (2019); arXiv:1907.02392 [cs.CV].
- [33] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, A. Rousselot, R. Winterhalder, L. Ardizzone and U. Köthe, Invertible networks or partons to detector and back again, *SciPost Phys.* **9** (2020) 074; arXiv:2006.06685 [hep-ph].
- [34] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman and T. Plehn, GAN-plinging event samples (2020); arXiv:2008.06545 [hep-ph].
- [35] G. P. Lepage, A new algorithm for adaptive multidimensional integration, *J. Comput. Phys.* **27**(2) (1978) 192.
- [36] M. Abadi *et al.*, Tensorflow: A system for large-scale machine learning (2016); arXiv:1605.08695 [cs.DC].
- [37] J. Bendavid, Efficient Monte Carlo integration using boosted decision trees and generative deep neural networks (2017); arXiv:1707.00028 [hep-ph].
- [38] M. D. Klimek and M. Perelstein, Neural network-based approach to phase space integration (2018); arXiv:1810.11509 [hep-ph].
- [39] S. Carrazza and J. M. Cruz-Martinez, VegasFlow: Accelerating Monte Carlo simulation across multiple hardware platforms, *Comput. Phys. Commun.* **254** (2020) 107376; arXiv:2002.12921 [physics.comp-ph].
- [40] E. Bothmann and L. Debbio, Reweighting a parton shower using a neural network: The final-state case, *J. High Energy Phys.* **01** (2019) 033; arXiv:1808.07802 [hep-ph].
- [41] F. Bishara and M. Montull, (Machine) learning amplitudes for faster event generation (2019); arXiv:1912.11055 [hep-ph].
- [42] S. Badger and J. Bullock, Using neural networks for efficient evaluation of high multiplicity scattering amplitudes (2020); arXiv:2002.07516 [hep-ph].
- [43] F. Chollet, GitHub repository (2015), <https://keras.io/>.
- [44] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2014); arXiv:1412.6980 [cs.LG].
- [45] B. Nachman, A guide for deploying deep learning in LHC searches: How to achieve optimality and account for uncertainty (2019); arXiv:1909.03081 [hep-ph].
- [46] G. Kasieczka, M. Luchmann, F. Otterpohl and T. Plehn, Per-object systematics using deep-learned calibration, *SciPost Phys.* **9** (2020) 089; arXiv:2003.11099 [hep-ph].
- [47] S. Bollweg, M. Haußmann, G. Kasieczka, M. Luchmann, T. Plehn and J. Thompson, Deep-learning jets with uncertainties and more, *SciPost Phys.* **8**(1) (2020) 006; arXiv:1904.10004 [hep-ph].
- [48] A. Butter, T. Plehn and R. Winterhalder, How to GAN event subtraction, *SciPost Phys.* **3** (2020) 009; arXiv:1912.08824 [hep-ph].
- [49] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, JUNIPR: A framework for unsupervised machine learning in particle physics, *Eur. Phys. J. C* **79**(2) (2019) 102; arXiv:1804.09720 [hep-ph].

- [50] T. Plehn, Lectures on LHC physics, *Lect. Notes Phys.* **886** (2015); arXiv:0910.4182 [hep-ph].
- [51] C. Sauer, Towards a data-driven simulation of QCD radiation with generative models utilizing machine learning methods, Heidelberg thesis (2019).
- [52] L. de Oliveira, M. Paganini and B. Nachman, Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis, *Comput. Softw. Big Sci.* **1**(1) (2017) 4; arXiv:1701.05927 [stat.ML].
- [53] DELPHES 3, J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens and M. Selvaggi, DELPHES 3, A modular framework for fast simulation of a generic collider experiment, *J. High Energy Phys.* **02** (2014) 057; arXiv:1307.6346 [hep-ex].
- [54] F. A. Dreyer, G. P. Salam and G. Soyez, The lund jet plane, *J. High Energy Phys.* **12** (2018) 064; arXiv:1807.04758 [hep-ph].
- [55] R. Verheyen and B. Stienen, Phase space sampling and inference from weighted events with autoregressive flows (2020); arXiv:2011.13445 [hep-ph].
- [56] M. Backes, A. Butter, T. Plehn and R. Winterhalder, How to GAN event unweighting (2020); arXiv:2012.07873 [hep-ph].
- [57] J. Alwall *et al.*, The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations, *J. High Energy Phys.* **07** (2014) 079; arXiv:1405.0301 [hep-ph].
- [58] B. Hashemi, N. Amin, K. Datta, D. Olivito and M. Pierini, LHC analysis-specific datasets with Generative Adversarial Networks, arXiv:1901.05282 [hep-ex].
- [59] J. Arjona Martinez, T. Q. Nguyen, M. Pierini, M. Spiropulu and J.-R. Vlimant, Particle Generative Adversarial Networks for full-event simulation at the LHC and their application to pileup description, arXiv:1912.02748 [hep-ex].
- [60] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro and E. Shelhamer, cudnn: Efficient primitives for deep learning, arXiv:1410.0759 [cs.NE].
- [61] R. Di Sipio, M. Faucci Giannelli, S. Ketabchi Haghighat and S. Palazzo, DijetGAN: A generative-adversarial network approach for the simulation of QCD dijet events at the LHC, *J. High Energy Phys.* **08** (2020) 110; arXiv:1903.02433 [hep-ex].
- [62] M. Cacciari, G. P. Salam and G. Soyez, FastJet user manual, *Eur. Phys. J.* **C72** (2012) 1896; arXiv:1111.6097 [hep-ph].
- [63] CMS, S. Chatrchyan *et al.*, Search for supersymmetry with razor variables in pp collisions at $\sqrt{s} = 7$ TeV, *Phys. Rev. D* **90**(11) (2014) 112001; arXiv:1405.3961 [hep-ex].
- [64] Y. Alanazi, N. Sato, T. Liu, W. Melnitchouk, M. P. Kuchera, E. Pritchard, M. Robertson, R. Strauss, L. Velasco and Y. Li, Simulation of electron-proton scattering events by a feature-augmented and transformed generative adversarial network (FAT-GAN) (2020); arXiv:2001.11103 [hep-ph].
- [65] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf and A. J. Smola, A kernel method for the two-sample problem (2018); arXiv:0805.2368 [cs.LG].

- [66] K. Kondo, Dynamical likelihood method for reconstruction of events with missing momentum. 1: Method and toy models, *J. Phys. Soc. Jap.* **57** (1988) 4126.
- [67] T. Martini and P. Uwer, Extending the matrix element method beyond the born approximation: Calculating event weights at next-to-leading order accuracy, *J. High Energy Phys.* **09** (2015) 083; arXiv:1506.08798 [hep-ph].
- [68] M. Kraus, T. Martini and P. Uwer, Matrix element method at NLO for (anti-) k_t -jet algorithms, *Phys. Rev. D* **100**(7) (2019) 076010; arXiv:1901.08008 [hep-ph].
- [69] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman and J. Thaler, OmniFold: A method to simultaneously unfold all observables, *Phys. Rev. Lett.* **124**(18) (2020) 182001; arXiv:1911.09107 [hep-ph].
- [70] K. Datta, D. Kar and D. Roy, Unfolding with generative adversarial networks (2018); arXiv:1806.00433 [physics.data-an].
- [71] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn and R. Winterhalder, How to GAN away detector effects, *SciPost Phys.* **8**(4) (2020) 070; arXiv:1912.00477 [hep-ph].

Part IV

Machine Learning Platforms

This page intentionally left blank

Chapter 8

Distributed Training and Optimization of Neural Networks

Jean-Roch Vlimant* and Junqi Yin†

**California Institute of Technology, Pasadena, CA 91125, USA
jvlimant@caltech.edu*

*†Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
yinj@ornl.gov*

Deep learning models are yielding increasingly better performances thanks to multiple factors. To be successful, model may have large number of parameters or complex architectures and be trained on large dataset. This leads to large requirements on computing resource and turn around time, even more so when hyperparameter optimization is done (e.g. search over model architectures). While this is a challenge that goes beyond particle physics, we review the various ways to do the necessary computations in parallel, and put it in the context of high-energy physics.

1. Introduction

The main aspects of distributed training have been recently well reviewed in [1] and we refer to it for a more in depth discussion on technical details. There exists a rich literature on distributed training of neural networks and notably [1–4], recommended as supplementary reading.

It is commonly agreed that deep learning has shown great success over the last decade, thanks to the creation of large labeled datasets, advancement in model architectures, and increase in computation power — in part due to general purpose graphical processing units (GPU). With ever growing complexity of datasets and models, and despite the acceleration provided by GPU, training can

still last for days and weeks on single device. Besides the training of a single model, it is often necessary to perform an optimization over some parameters, that are otherwise not learnable with gradient descent.

With the acceleration of the adoption of deep learning in high-energy physics [5–8], it becomes necessary to look at ways to reduce the effective training time. While most simple neural network models and other classical machine learning methods can be trained in reasonable time, more advanced models like graph neural network (Chapter 12) and generative adversarial network (Chapters 6 and 7) can be hard to train [9]. Complex architectures that exhibit large training time per epoch are often just discarded solely due to the time it would take to bring them to converge — let alone doing hyperparameter tuning. Improvement of the time to solution is required to make the development of such models more amenable. Distributed training may reduce weeks of training down to days.

It should be noted that the challenge of accelerating the time to convergence is not specific to high-energy physics (HEP). However, the specific computing and software environment of HEP might limit the possibilities otherwise available. For example, due to budget constraints, it is not given that GPUs are available for training. Furthermore, the software options are limited by the requirement of affordable long-term support.

We provide in this chapter a description of the key aspects of distributed training and optimization as a practical guide to developing large models with large amount of data. This chapter is organized as follows. After introducing the formalism of training and optimizing neural network models in Sec. 2, highlighting the possible strategies to parallelize computation, we start in Sec. 3 with the parameter distribution strategy, which was the first to be adopted as a way to speed up the training of models. We then describe in Sec. 4 the data distributed strategy that seems to be widely adopted currently, thanks to its ease of use. We go over model parallelism in Sec. 5 and recent development in generic deployment of this otherwise complicated method to implement. In Sec. 6, we get into the details of model search and optimization, key to the success of developing

high-performance deep learning applications. We conclude in Sec. 7 with an overview of advancements of software and outlooks on distributed training.

2. Neural Network Optimization Formalism

Neural network models can generally be represented as a function $f_{\theta,h}(X)$ predicting some quantity Y . Here X is some input, θ and h are parameter vectors. A loss function $\mathcal{L}_{\theta,h} := F(f(X))|_{\theta,h}$ is defined to characterize the fitness of the model to a specific task — for example, the binary cross entropy can be used in a binary classification supervised task. In the general setup, one is searching for the two sets of parameters θ^* and h^* , which provide an optimal value over some data. We distinguish the parameters h from θ by the fact that \mathcal{L} is differentiable with respect to θ , but not with respect to h — h are the so-called *hyperparameters*.

By definition, θ^* can be approximated by using the gradient descent method-based gradient of the loss function \mathcal{L} with respect to the model parameters θ , noted:

$$\nabla_{\theta}\mathcal{L}_{\theta,h} = \nabla F|_{f_{\theta,h}} \cdot \nabla_{\theta}f_{\theta,h} \quad (1)$$

starting from an initial set of parameters $\theta_{h,0}$ that may depend on h . We note that some models and loss definitions involve gradient ascent instead of gradient descent and the methods described in this chapter are all applicable in this situation. In practice, *stochastic gradient descent* (SGD) is found to be more stable than using individual gradients to update the model parameters. In this algorithm, the gradient is averaged over subsets $\{X_{i_b}\}$ indexed by b — referred to as *batch* therein, but also called *mini-batch* in the literature — of the whole training dataset. In such case, the gradient for each individual input can be computed in parallel before calculating the average over a batch as

$$\nabla_{\theta}\mathcal{L}|_b := \mathbf{E}_{\{X_{i_b}\}} [\nabla_{\theta}\mathcal{L}_{\theta,h}]. \quad (2)$$

This is the key ingredient that makes the computation of SGD efficient on GPU as it is under single program multiple data (SPMD)

programming style. During the optimization procedure, the parameter θ is updated with the rule

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}|_b, \quad (3)$$

where η is either fixed or dynamic and referred to as the *learning rate*. Equation (3) can be written more generically

$$\theta \leftarrow \theta - \Psi(h, \nabla_{\theta} \mathcal{L}|_b), \quad (4)$$

where $\Psi(., x) = \eta x$ in its simplest form of Eq. (3). An *epoch* is the cycle of the algorithm during which the whole training dataset — all batches — is used once to update the model parameters. It typically requires numerous epochs to train a model properly. It should be noted that the typical neural network loss function is non-convex and therefore optimization with gradient descent can be subject to being stuck in local minima, preventing the procedure to reach the absolute global minimum — which might actually be degenerate. Obviously, one may compute the effective gradient over a given batch by splitting the computation over multiple parts — or *shards* — of a batch. This is the key aspect of data distribution to which we come back in Sec. 4. Additionally, one may compute the effective gradient of several batches in parallel, with the caveat that applying the update rule (Eq. (3)) is not exactly possible (see Sec. 3).

The case of generative adversarial networks (GAN) brings in some additional complication because of the presence of two models and a training procedure that alternates between updating the parameters of the *discriminator/critic* and the *generator*. We come back to these points in Sec. 3.

In the case where the model is composed of multiple layers L_l indexed by l , i.e. $f_{\theta,h} := L_{N_L}(\cdots L_1(L_0(X))\cdots) := [\circ_{l=0}^{N_L} L_l](X)$, the derivation chain rule applies and we obtain

$$\nabla_{\theta_i^m} f = \left\{ \prod_{l=N_L}^{m+1} \nabla L_l|_{A_{l-1}(X)} \right\} \nabla_{\theta_i^m} L_m|_{A_{m-1}(X)}, \quad (5)$$

where θ_i^m is a parameter of layer L_m . We introduced the notation $A_m(X) = [\circ_{l=0}^m L_l](X)$ for the *activation* of layer m . The *forward pass*

is the computation of the value of successive layers using activation values for preceding layers. *Back propagation* is the computation of the gradients for layer L_m using gradients already computed for any layers $l > m$.

In multiple occasions, the computation required for a single layer L_l and its Jacobian ∇_{L_l} involve tensor products that allow for some level of parallelism. Convolutional layers for example can have the computation of each filter on each patch of the input image be processed on separate devices. We consider this as model parallelism and come back to this point in Sec. 5.

Assuming an arbitrary layer separation S such that $m \leq S < N_L$, the right-hand side term of Eq. (5) factorizes as

$$\left[\left\{ \prod_{l=N_L}^{S+1} \nabla_{L_l} \Big|_{[\mathbf{c}_{k=S+1}^{l-1} L_k](A_S(X))} \right\} \right] \\ \times \left[\left\{ \prod_{l=S}^{m+1} \nabla_{L_l} \Big|_{A_{l-1}(X)} \right\} \nabla_{\theta_i^m} L_m \Big|_{A_{m-1}(X)} \right]. \quad (6)$$

The first bracketed product (referred to as B_S) requires only one input involving the layers $m \leq S$: the term $A_S(X)$. Furthermore, B_S appears in the calculation of the gradients for all layers $m \leq S$. Therefore, the calculations of the gradients for the set of layers $L_{m>S}$ and the set of layers $l_{m \leq S}$ only require knowledge of the two disjoint sets of layers, respectively, once A_S (during the forward pass) and B_S (during back propagation) have been computed. Hence S can be chosen to balance memory utilization for example. This is a key aspect of model parallelism that we expand on in Sec. 5.

The optimal value h^* needs to be found with other optimization methods — notably Bayesian optimization and evolutionary algorithms. We note that if all values of h are continuous, h^* can still be found with gradient descent, using numerical evaluation of the gradient — with a finite difference method for example. The discussion on *hyperparameter optimization* (HPO) in this chapter is unchanged if one decides to use any other non-differentiable function as *figure of merit*: $\mathcal{F}_{\theta,h}(\{X_i\})$, instead of using \mathcal{L} . By extension, we include in the

hyperparameter set, all parameters used towards obtaining the optimal set of parameters for the model — such as learning rate, model initialization parameters, batch size. The optimal model parameters obtained at the optimal value of the hyperparameters h^* is noted θ^* .

The full dataset is often divided into three independent subsets: *training*, *validation* and *testing* sets. The search for θ^* (*training* procedure) is done on the training set, the validation set is used to estimate the generalization performance, and the testing set is used solely to report a final performance of the optimal model. The search for h^* (HPO procedure) is done by optimizing $\mathcal{F}_{\theta,h}$ over the validation set, and provides further ways in which the computation may be done in parallel. The *K-folding* procedure is recommended when comparing model performance as it provides a better estimation of the mean and variance of the performance. We come back to the process of hyperparameter optimization using K-folding in Sec. 6.

3. Parameter Distribution

As explained in Sec. 2, the calculation of the gradient averaged over a batch can be done for multiple batches in parallel. The caveat is that the gradients are calculated from a given set of model parameters and not necessarily applied to update the same model parameters. To be more precise, given two batches b_1 and b_2 , in the gradient update done sequentially, we have

$$\begin{aligned}\theta_1 &\leftarrow \theta_0 - \eta \nabla_{\theta} \mathcal{L}|_{b_1, \theta_0}, \\ \theta_2 &\leftarrow \theta_1 - \eta \nabla_{\theta} \mathcal{L}|_{b_2, \theta_1},\end{aligned}\tag{7}$$

while we obtain

$$\begin{aligned}\theta_1 &\leftarrow \theta_0 - \eta \nabla_{\theta} \mathcal{L}|_{b_1, \theta_0}, \\ \theta_2 &\leftarrow \theta_1 - \eta \nabla_{\theta} \mathcal{L}|_{b_2, \theta_0}\end{aligned}\tag{8}$$

in the case of computing the gradients concurrently and applying them sequentially. On the second update in Eq. (8), the gradients are computed on b_2 , from θ_0 , but applied on θ_1 — different than θ_0 . This results in *staleness of gradients* and a slowdown in convergence of the models [10, 11]. The benefit is that the time to run a full

epoch linearly decreases with the number of batches processed in parallel. The cost is that the decrease of the loss after a fixed number of epochs is not necessarily better when increasing the number of batches processed in parallel.

Multiple proposals have been made to mitigate the effect of outdated gradients, mostly resulting in changing the value of η . Having η be a matrix in an element-wise multiplication as in the *adagrad* algorithm [12] provides improvements in convergence [2]. Alternatively, inspired from a physics concept, the *gradient energy matching* algorithm [13] offers a provable way of stabilizing convergence of the distributed training.

It should be noted that the mode in which the gradients — that are computed in parallel — are used to update the model parameters is subject to contention. Indeed a bottleneck occurs when the effective time for gradients to be computed is smaller than the update time of the model parameters, if there are many parallel processes for instance. This underlines the fact that the parameter strategy cannot scale to an infinite number of parallel processes. One solution to this problem is to create a multi-level [11] (as opposed to binary level so far) hierarchy of processes that reports updates up one level at a time, effectively reducing the single-updating-process bottleneck, at the cost of increasing the staleness of gradients.

As mentioned in Sec. 2 and Chapters 6 and 7, GAN are composed of two models that are trained in an alternate manner, and possibly with different number of parameter updates at each cycle. The parameters can be synchronized either after both models were trained on a full batch, or after each parameter update, or each model. The former is the strategy adopted in [9, 14, 15]. The latter would be impacted even more strongly by the bottleneck created in updating the parameters centrally.

Take Home Message. The method to parallelize training using the parameter distribution strategy is intrinsically limited in scalability due to a combination of the centrality of the parameter update and staleness of the gradients. Training GAN under such strategy is furthermore complicated by the presence of two models with

different training dynamics. The mild acceleration obtained with parameter distribution can be complemented with other means of parallelization — as we will see the next sections of this chapter.

4. Data Distributed Training

The data distributed training follows the SPMD paradigm and is the most widely adopted approach to the distributed deep learning due to several advantages that it can offer: straightforward to implement, model agnostic to apply and generally efficient to scale up. In a typical data parallel paradigm flow chart as shown in Fig. 1, a model is replicated on each device and the forward pass on different shards of a batch are calculated independently following Eq. (2). The gradients of all processes are synchronized after the back propagation, via the *allreduce* collective communication. As the parallelism increases, the effective batch size hence is linearly proportional to the total number of processes (model replicas). The forward model, backward model, and gradients calculations can often be performed in half precision operations (fp16), and are marked in mixed colors in the flow chart of a typical mixed precision training. The *allreduce* is usually performed in single precision to avoid gradient overflow,

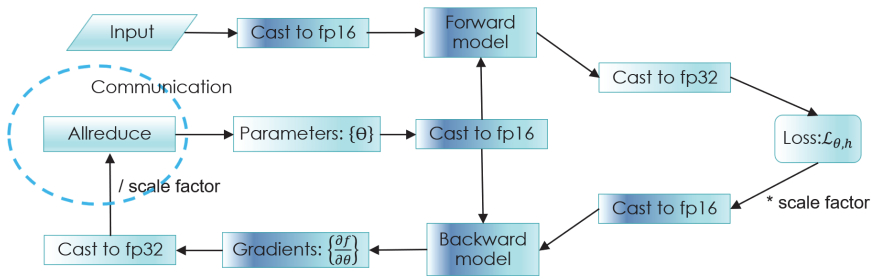


Fig. 1. Typical processing flow chart in a data distributed training, where the potential half precision operations (fp16) are marked with mixed colors. The communication is invoked after gradients are estimated from backward model, and the parameters of the model are then updated (Eq. (4)) with synchronized gradients. The scale factor is needed to avoid the numerical overflow during the $\text{fp16} \leftarrow \text{fp32}$ conversion.

Table 1. Framework built-in data parallel support. MirroredStrategy and MultiWorkerMirroredStrategy are equivalent to DataParallel and DistributedDataParallel, respectively.

Framework\Distribution	Single node	Multi-node
TensorFlow	MirroredStrategy	MultiWorkerMirroredStrategy
PyTorch	DataParallel	DistributedDataParallel

but fp16 is also supported by most up-to-date libraries. With neural network like ResNet50, where the message size per model replica is about 100 MB, the allreduce communication can soon become a bottleneck when training at scale. Based on the novel ring-allreduce algorithm, Horovod [16] is a high performance communication library for data distributed training. The algorithm is network bandwidth optimized, and each process sends and receives gradient messages $2(n-1)/n$ times, where n is the number of processes. It can therefore scale efficiently for large n since the total message size communicated per process becomes a constant when $n \rightarrow \infty$. Given the success of ever-larger neural network models and datasets, popular deep learning frameworks also provide built-in support for data parallel training. The commonly used data-parallel methods in TensorFlow and PyTorch are listed in Table 1. The advantages of using these methods for distributed training are compatibility (integrated with the framework), user-friendliness and performance (give the support for high performing communication fabrics). The usage mode (API) is however not flexible enough to support novel communication patterns and mostly limited to data parallelism only.

One common pitfall for data parallel training is that model accuracy deteriorates at large batch sizes. It is argued [17] that the estimation of the gradient at each step is more accurate with larger batches and hence the optimization becomes smoother — i.e. with less stochasticity —, resulting in a higher probability to be trapped in a local minima. There are strategies to mediate this effect such as layer-wise adaptive rate scaling (LARS) [18], but in general the upper bound for batch size is still limited around $64K$ for most applications

based on first-order optimization (e.g. SGD). Natural gradient methods can push the boundary of the large-batch training further [19] at the cost of expensive evaluations of second-order derivatives.

Another pitfall is that batch normalization layers [20] are not effective when the shard size is too small, since there are not enough samples to obtain sufficiently accurate mean and variance of the layer activation values. The synchronization of such layer is not commonly implemented in data parallel libraries (e.g. Horovod, PyTorch Data-Parallel, etc.). The models with batch normalization layer (widely used after a convolution layer to improve regularization and accuracy), trained with small shard size — i.e. many processes — perform significantly worse in both convergence and accuracy, than that of a model trained without data parallelism. Therefore, the optimal batch for data-parallel training is not simply the optimal batch size for single replica divided by total number of processes; it depends on both the data and the model.

With high-performance communication fabric (e.g. InfiniBand) and software (e.g. NCCL enabled GPU Direct RDMA), data parallelism can achieve excellent scaling efficiency on HPC platforms. Figure 2 shows an Exascale data-parallel deep learning application [21] on the Summit supercomputer, which tackles a long-standing inverse problem in atomic imaging (a.k.a phase problem) by deep learning reconstruction of the electron densities from microscopic images. About 93% scaling efficiency has been achieved up to 27,000 GPUs with a peak performance of 2.1 EFLOPS in half precision.

Take Home Message. Data-parallel training is the most popular distributed training methods thanks to it being model agnostic (CNN, RNN, or GAN) and simple to implement (framework built-ins, see Table 1, or third party library plugins, e.g. Horovod). On the other hand, for applications with very a large model, a model may not be able to fit into a single device. For application with very large input data, model accuracy is likely to suffer at large batch size. For those use cases, parallelism beyond data parallel paradigm needs to be explored, which will be discussed in the next section.

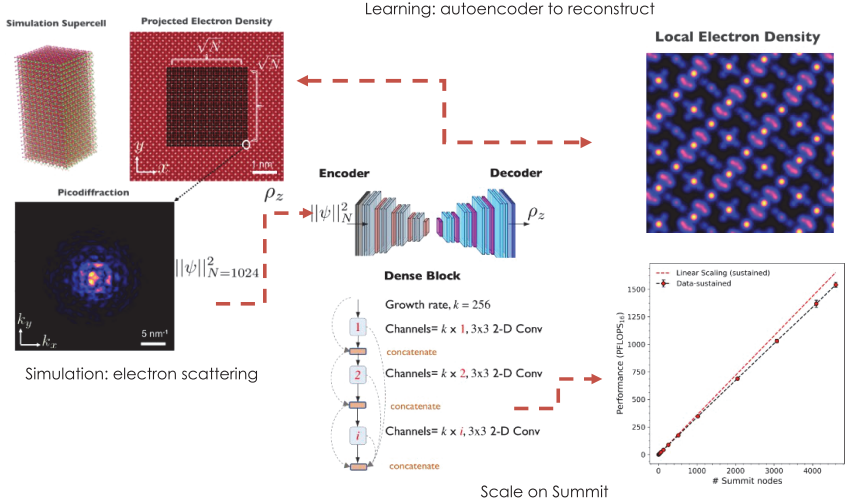


Fig. 2. An example [21] of Exascale data-parallel deep learning on the Summit supercomputer, which consists of both a simulation component that generates electron microscopic data, and a learning component that trains to reconstruct the local electron density.

5. Model Parallelism

As illustrated in Sec. 2, another level of parallelism that can be explored is at the model level. This is useful when a model is too big to fit into the GPU memory such as state-of-the-art NLP models with billions of parameters [25], or when the input data is of very high dimension such as medical [27], geospatial imaging or particle tracking [28]. As shown in Table 2, the size of popular models has grown from millions to billions of parameters in just over 5 years.

Given the execution dependency between layers of a neural network, simple device placement of different layers will not execute at the same time. There are two common implementations for model parallelism. First, *pipelining*, e.g. GPipe [29], where a batch is divided into micro-batches and then different micro-batches at different layers can execute in parallel albeit at a reduced efficiency. Second, *tensor contraction*, e.g. Mesh-TensorFlow [30], where other dimensions of a typical input batch can be parallelized in addition to the batch dimension.

Table 2. Evolution of model size and parallel framework, from data parallel training on computer vision models (typically of millions of parameters) to hybrid parallel training on large language models (up to billions of parameters).

Model	Year	Number of parameters (billion)	Parallel framework
ResNet50 [22]	2015	0.025	PyTorch data parallelism
BERT-Large [23]	2018	0.34	TensorFlow data parallelism
GPT-2 [24]	2019	1.5	N/A
Megatron-LM [25]	2019	8.3	PyTorch hybrid parallelism
T-NLG [26]	2020	17	PyTorch hybrid parallelism

In comparison, the GPipe approach is more generic for sequential models but it suffers from low-scaling efficiencies due to the so-called *bubble* [29], namely the idle time caused by the sequential dependency of the execution of a micro-batch among model layers, and the same large batch training issue as in data parallelism. On the other hand, the Mesh-TensorFlow method can be more efficient but it relies on customizing each operation in the model.

In addition to augmenting parallel capabilities on top of existing popular frameworks such as TensorFlow and PyTorch, several notable efforts tackle the same issue from the traditional HPC perspective, including the Livermore big artificial neural network toolkit (LBANN [31]), which provides model parallel acceleration via domain decomposition, and a Legion [32] based framework that uses task graph parallel strategy [33]. Both domain decomposition and task graph are generic parallel computing techniques that go beyond distributed training and the scope of this chapter. These developments have seen adoptions typically in the HPC community.

In practice, the model parallelism alone usually will not scale due to the cross-node communication latency. Therefore, a hybrid scheme (see Fig. 3 and Table 2) with data parallelism on the batch dimension and model parallelism on other dimensions strikes a better balance between scaling efficiency and model capabilities (larger model, faster convergence, etc) than that of data or model parallelism alone.

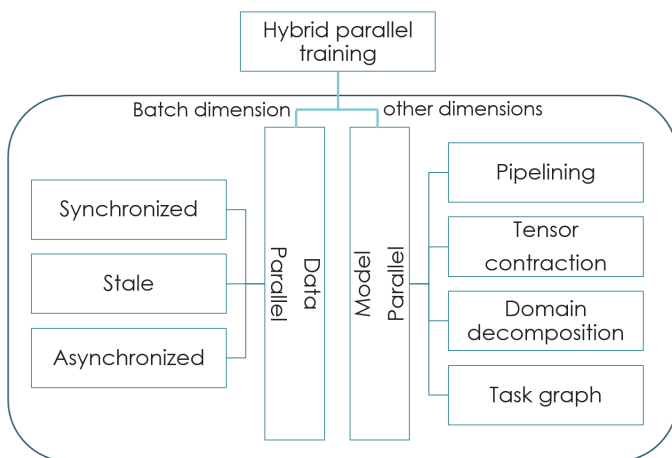


Fig. 3. Hybrid parallel training scheme: data parallelism applied on the batch dimension and model parallelism on others. For data parallelism, depending on how often models are synchronized, it can be every step (synchronized), every few steps (stale), or asynchronous. For model parallelism, there are different implementations including pipelining [29], tensor contraction [27], domain decomposition [31], and task graph [33].

Take Home Message. Model parallelism is becoming more important as the model size and the input dimension grow. It further improves the distributed training efficiency at the cost of software complexity. So far, there is no common framework for model parallelism in the community and most developments are model specific. Model parallelism can be combined with the other means of parallelism presented so far and is also fully orthogonal to hyperparameter optimization discussed in the next section.

6. Hyperparameter Optimization

As introduced in Sec. 2, the training procedure of a model, to obtain an optimal set of parameters θ^* may depend on a set of hyperparameters h . The optimal value h^* that yields the best of the optimal model parameters can be found by optimization of a figure of merit \mathcal{F} — usually the loss \mathcal{L} (see Sec. 2).

When comparing the performance of two models obtained from two different sets h_1 and h_2 , it is important to control whether the difference observed is relevant and significant. If the training is done once on a training set from a unique set of initial parameters, and the model performance is evaluated on a fixed testing set, then there is an uncertainty related to these choices. The cross validation procedure provides a way to address this uncertainty and in particular to obtain a better estimator of the model performance. The training set is split in K parts — or *folds* — $\{F_k\}$. K train/test pair can therefore be formed

$$\left(\{X_i\}_{i \notin F_k}, \{X_i\}_{i \in F_k} \right)_k, \quad k = 1, \dots, K. \quad (9)$$

Running K trainings one obtains a set of K estimations of the figure or merit: $\{\mathcal{F}_k\}$. The performance estimator $\overline{\mathcal{F}} = \mathbf{E}_k[\mathcal{F}_k]$ provides a better way to compare models trained under different settings. The variance of the performance is an indicator of the instability of the model training when changing random seeds and training partition. This might be considered also as a selection criterion for the model. It should be noted that training over the K folds can be done fully in parallel, and therefore offer an almost perfect scaling. Residual inefficiency can arise if the training over one fold requires significantly larger amount of resources than the others.

In most search strategies of the optimal hyperparameters, the trainings with different parameter sets h_i are done independently of each other and therefore offer another level of parallelism. However, large fluctuation of resource needs can be expected for different parameter sets and some scaling bottlenecks may arise. Other strategies [34] involve cross-talk between hyperparameter sets, introducing some level of interlocking of processes, with the benefit of an improved search mechanism.

Strategies have been proposed for the optimization of hyperparameters that do not involve the architectural aspect of the model (number of layers, neurons, etc.). In [34], the model parameters and hyperparameters are evolved at the same time over a population of models, effectively learning a scheduling of changes of hyperparameters that favors the convergence of the model. Conversely, some

strategies concentrate only on the hyperparameters driving the architecture of the model. In [35], the architecture search starts with a simple model, and new layers are added in steps, on top of the best performing previous layer setup. This reduces the navigation of the large phase space of model architectures in which both the number of layers and number of neurons per layer are varied simultaneously, to a subset where only the size of the last layer is modified at once.

Strategies that aim at optimizing a generic set of hyperparameters most commonly use a Bayesian optimization [36] or an evolutionary algorithm approach [37, 38] as an efficient alternative to a simple grid search over the large space of hyperparameters. Bayesian optimization with a Gaussian process (GP) assumption on the prior of the hyperparameters is adequate if the prior is sufficiently regular to be modeled with a GP. The optimization process is rather sequential — the choice of a new set to evaluate is conditional to evaluation of all previous choices — but some level of parallelism can be introduced to favor exploration. The GP approximation can become computationally prohibitive if the number of steps is too large — when the size of the hyperparameters space is large for example. The Bayesian approach is favorable when the user is limited in the number of usable parallel processes — such as when making use of a small-scale institutional cluster, or personal computer. Evolutionary algorithms, on the other hand, proceed with evaluating the performance (or fitness) of hyperparameters sets in parallel — as individuals of a population — and synchronize all computing processes during population mutation and breeding. This approach can offer a better exploration of hyperparameters, at the cost of more parallel resources — available as such on high performance computing (HPC) centers and large-scale clusters.

Take Home Message. Hyperparameter search is an optimization problem on top of the model parameter optimization. A simple grid search algorithm becomes quickly prohibitive with the number of hyperparameters. Bayesian optimization may be recommended for an intermediate number of hyperparameter, on intermediate scale computing cluster. Evolutionary algorithms are adapted for large

hyperparameter space, and large-scale computing clusters, such as HPC facility.

7. Summary and Discussion

We have succinctly introduced the possible ways of running the computation for model training and optimization in parallel. We reviewed the latest developments of each strategy, their strengths and weaknesses. The otherwise highly nested loop involved in model training, K-folding, and hyperparameter optimization, can be largely unfolded in parallel computations. In the following, we provide remarks on key aspects of distributed training, and prospects in the field of particle physics.

Communication bottleneck. Similar to performance optimization of HPC applications, improving the scaling of distributed training is about finding the balance point between compute, I/O, and communication. Regardless of whether one uses data, model, or hybrid parallelism, most deep neural network applications can benefit from performance boost of the underlying communication protocol, since many such applications become network-bound at large scale (e.g. the message size is 100 MB and 1.44 GB for ResNet50 and BERT-large, respectively). It is reported [39] that model accuracy can be maintained while the message size for gradients is being reduced by up to 2 order of magnitude via a combination of compression techniques (e.g. clipping, sparsification, accumulation, quantization).

Trading memory for computation. Since neural network training is usually performed on GPU devices and GPU memory is still a scarce resource, trading memory for computation is sometimes a necessary workaround. It however requires significant engineering effort, similarly to implementing model parallelism. One strategy is *memory check-pointing* [40], where only selected forward nodes in the computation graph are check-pointed while others are re-computed during the backward propagation for the gradient evaluation. This results into a $O(\sqrt{n})$ memory usage of a neural network model with n nodes.

Another strategy is via GPU memory management [41], where inactive tensors are automatically transferred from GPUs to the host and vice versa. This is transparent to users and the added performance penalty is often tolerable. As reported in [42], a specific random sampling of the computation graph during the back-propagation phase allows for a reduced memory burden in computing the gradients.

Locality of resources. Depending on the resources, the optimal distributed training strategy may vary. Computing nodes in HPC resources are often homogeneous and equipped with high-performance interconnects, where the ring allreduce communication usually works the best but only half of the theoretical network bandwidth is achievable (see Sec. 4). Cloud resources, on the other hand, can be provisioned and are more suitable for a parameter-server distribution strategy. BytePS [43] is one such example to take advantage of cloud resources, and is reported to perform better than the collective communication approach in a specific hardware setup, where total bandwidth of server nodes is no less than that of worker nodes. How to efficiently apply a similar parameter distribution approach to node-homogeneous HPC resources is not yet clear.

In most of the discussion in this review, the input dataset is assumed to be located in local storage. No consideration of data caching on the node or on the edge was done. However, in particle physics, the available data may be distributed over multiple regions, and hence a specific data management protocol may be required. The training computation could be done where the data is — with implications on the software framework — or the data would need to be moved and cached on the edge of the computing resource — with implications on the data management system.

Running the models. The software used in particle physics to run model in production is actually not dictated by machine learning considerations, and restrictions might arise in how the models are encoded. This requirement for compatibility of software might impose constraints on the framework used for distributed training, unless cross-framework conversion tools are kept extremely efficient.

Even in situations where the model is ran on remote platform [44, 45], therefore loosening some of the software compatibility constraints, the model components might have to be post-processed in particular ways to fit on the remote hardware — in particular on FPGAs. As a consequence, some of these inference-oriented post-processing steps (pruning, quantization, etc.) might need to be taken into account during the distributed training phase to ensure a maximally efficient algorithm in the end.

Framework consideration. In terms of software availability for distributed training, parameter and data parallelism are well supported both by frameworks (see Table 1) and third-party plugins, e.g. Horovod [16] for data parallelism, BytePS [43] for parameter distribution, etc. For model parallelism, on the other hand, libraries are still in early stages, e.g. Mesh-TensorFlow [30], PyTorch-GPipe [46], etc. There are also several hyperparameter optimization libraries, like Ray Tune [47]. What is lacking is a unified open software framework that can effectively explore all the parallelism in distributed training, and hence allow for an efficient use of computing resource and person power.

A particular aspect in particle physics is that budget constraints often lead research groups to develop their own software so that the support model stays within the community. Although the problem of distributed training is not specific to high-energy physics, some of the features could be specific, as explained above, and need sustainable support. It is therefore of the utmost importance to have good feature support and maintenance in the *framework-to-be-used* in particle physics.

Final Remarks. As the high-performance computing machines are entering the exascale era, distributed training and optimization of neural networks is one of the major areas that can harness this tremendous computing power. While there are many efforts exploring parallelisms from parameter distribution, data and model parallelism, to hyperparameter optimization, the community is still in need of an overarching solution. One noteworthy work in this direction is a framework [33] based on the task graph, a parallel

programming paradigm that is suitable for exascale. It can naturally combine the aforementioned distributed training strategies and hence enable an efficient end-to-end training workflow, although feature developments are lagging behind mainstream frameworks.

Acknowledgments

We thank authors of other chapters, Vlad Oles and Feiyi Wang for feedback on our manuscript. J.Y. is supported by the U.S. Department of Energy, Office of Science, National Center for Computational Sciences. This research used resources of the Oak Ridge Leadership Computing Facility, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. J.-R.V. is partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement no. 772369) and by the U.S. Department of Energy, Office of Science, Office of High Energy Physics under award numbers DE-SC0011925, DE-SC0019227 and DE-AC02-07CH11359.

References

- [1] T. Ben-Nun and T. Hoefler, Demystifying parallel and distributed deep learning: An in-depth concurrency analysis, *ACM Comput. Surv.* **52**(4) (2019) 65; doi: 10.1145/3320060.
- [2] J. Dean, *et al.*, Large scale distributed deep networks, in *Proc. 25th Int. Conf. Neural Information Processing Systems — Volume 1, NIPS'12*, (Curran Associates Inc., Red Hook, NY, 2012), pp. 1223–1231.
- [3] J. Yin, S. Gahlot, N. Laanait, K. Maheshwari, J. Morrison, S. Dash and M. Shankar, Strategies to deploy and scale deep learning on the summit supercomputer, in *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)* (2019), pp. 84–94; doi: 10.1109/DLS49591.2019.00016.
- [4] T. Kurth, Tensorflow at scale MPi, RDMA and all that, in *Proc. Cray User Group 2018*, Stockholm (2018).
- [5] A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, A. Himmel, A. Aurisano, K. Terao and T. Wongjirad, Machine learning at the energy and intensity frontiers of particle physics, *Nature* **560**(7716) (2018) 41–48; doi: 10.1038/s41586-018-0361-2.

- [6] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto and L. Zdeborová, Machine learning and the physical sciences, *Rev. Mod. Phys.* **91**(4) (2019) 045002; doi: 10.1103/RevModPhys.91.045002.
- [7] D. Bourilkov, Machine and deep learning applications in particle physics, *Int. J. Mod. Phys. A* **34**(35) (2020) 1930019; doi: 10.1142/S0217751X19300199.
- [8] iML. A living review of machine learning for particle physics. URL <https://iml-wg.github.io/HEPML-LivingReview/> (2020).
- [9] S. Vallecorsa, D. Moise, F. Carminati and G. R. Khattak, Data-parallel training of generative adversarial networks on hpc systems for hep simulations, in *2018 IEEE 25th Int. Conf. High Performance Computing (HiPC)* (2018) pp. 162–171; doi: 10.1109/HiPC.2018.00026.
- [10] S. Zhang, A. Choromanska and Y. LeCun, Deep learning with elastic averaging sgd, in *Proc. 28th Int. Conf. Neural Information Processing Systems — Volume 1, NIPS’15* (MIT Press, Cambridge, MA, 2015), pp. 685–693.
- [11] D. Anderson, J. Vlimant and M. Spiropulu, An MPi-based python framework for distributed training with Keras (2017); arXiv:1712.05878.
- [12] J. Duchi, E. Hazan and Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, *J. Mach. Learn. Res.* **12** (2011) 2121–2159.
- [13] J. Hermans and G. Louppe, Gradient energy matching for distributed asynchronous gradient descent (2018); arXiv:1805.08469.
- [14] S. Vallecorsa, F. Carminati, G. Khattak, D. Podareanu, V. Codreanu, V. Saletore and H. Pabst, Distributed training of generative adversarial networks for fast detector simulation, in *High Performance Computing*, eds. R. Yokota, M. Weiland, J. Shalf and S. Alam (Springer International Publishing, Cham, 2018), pp. 487–503.
- [15] J.-R. Vlimant, F. Pantaleo, M. Pierini, V. Loncar, S. Vallecorsa, D. Anderson, T. Nguyen and A. Zlokapa, Large-scale distributed training applied to generative adversarial networks for calorimeter simulation, *EPJ Web Conf.* **214** (2019) 06025; doi: 10.1051/epjconf/201921406025.
- [16] A. Sergeev and M. D. Balso, Horovod: fast and easy distributed deep learning in TensorFlow (2018); arXiv:1802.05799.
- [17] S. McCandlish, J. Kaplan, D. Amodei and O. D. Team, An empirical model of large-batch training (2018); arXiv:1812.06162.
- [18] Y. You, I. Gitman and B. Ginsburg, Scaling SGD batch size to 32k for imagenet training (2017); arXiv:1708.03888.
- [19] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, R. Yokota and S. Matsuoka, Large-scale distributed second-order optimization using Kronecker-factored approximate curvature for deep convolutional neural networks, in *2019 IEEE/CVF Conf. Computer Vision and Pattern Recognition (VPR)*, Long Beach, CA (2019) pp. 12351–12359; doi: 10.1109/CVPR.2019.01264.
- [20] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015); arXiv: 1502.03167.
- [21] N. Laanait, J. Romero, J. Yin, M. T. Young, S. Treichler, V. Starchenko, A. Borisevich, A. Sergeev and M. Matheson, Exascale deep learning for scientific inverse problems (2019); arXiv:1909.11150.

- [22] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (2016) pp. 770–778; doi: 10.1109/CVPR.2016.90.
- [23] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding (2019); arXiv:1810.04805.
- [24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever, Language models are unsupervised multitask learners (2018). URL <https://d4mucfpkwywv.cloudfront.net/better-language-models/language-models.pdf>.
- [25] M. Shoenberger, M. A. Patwary, R. Puri, P. LeGresley, J. Casper and B. Catanzaro, Megatron-LM: Training multi-billion parameter language models using model parallelism (2019); arXiv:1909.08053.
- [26] S. Rajbhandari, J. Rasley, O. Ruwase and Y. He, Zero: Memory optimization towards training a trillion parameter models (2019); arXiv:1910.02054.
- [27] L. Hou, Y. Cheng, N. Shazeer, N. Parmar, Y. Li, P. Korfiatis, T. M. Drucker, D. J. Blezek and X. Song, High resolution medical image analysis with spatial partitioning (2019); arXiv:1909.03108.
- [28] S. Farrell *et al.*, Novel deep learning methods for track reconstruction, in *4th Int. Workshop Connecting The Dots 2018* (2018).
- [29] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le and Z. Chen, Gpipe: Efficient training of giant neural networks using pipeline parallelism (2018); arXiv:1811.06965.
- [30] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi and B. Hechtman, Mesh-TensorFlow: Deep learning for supercomputers, in *Advances in Neural Information Processing Systems 31*, eds. S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett (Curran Associates, Inc., 2018), pp. 10414–10423. URL <http://papers.nips.cc/paper/8242-mesh-tensorflow-deep-learning-for-supercomputers.pdf>.
- [31] N. Dryden, N. Maruyama, T. Benson, T. Moon, M. Snir and B. Van Essen, Improving strong-scaling of cnn training by exploiting finer-grained parallelism, in *2019 IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, (2019), pp. 210–220.
- [32] M. Bauer, S. Treichler, E. Slaughter and A. Aiken, Legion: Expressing locality and independence with logical regions, in *SC '12: Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis* (2012), pp. 1–11.
- [33] Z. Jia, M. Zaharia and A. Aiken, Beyond data and model parallelism for deep neural networks, in *Proc. Conf. Systems and Machine Learning (SysML), SysML'19, USA* (2019). URL <https://www.sysml.cc/doc/2019/16.pdf>.
- [34] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando and K. Kavukcuoglu, Population based training of neural networks (2017); arXiv:1711.09846.
- [35] M. L. Pasini, J. Yin, Y. W. Li and M. Eisenbach, A scalable constructive algorithm for the optimization of neural network architectures, *Parallel Computing* **104–105** (2021) 102788.

- [36] J. Snoek, H. Larochelle and R. P. Adams, Practical Bayesian optimization of machine learning algorithms (2012); arXiv:1206.2944.
- [37] C. D. Schuman, J. S. Plank, A. Disney and J. Reynolds, An evolutionary optimization framework for neural networks and neuromorphic architectures, in *2016 Int. Joint Conf. Neural Networks (IJCNN)* (2016) pp. 145–154; doi: 10.1109/IJCNN.2016.7727192.
- [38] S. R. Young, D. C. Rose, T. Johnston, W. T. Heller, t. P. Karnowski, T. E. Potok, R. M. Patton, G. Perdue and J. Miller, Evolving deep networks using HPC, in *Proc. Machine Learning on HPC Environments* (2017), 7; doi: 10.1145/3146347.3146355.
- [39] Y. Lin, S. Han, H. Mao, Y. Wang and W. Dally, Deep gradient compression: Reducing the communication bandwidth for distributed training (2018); arXiv:1712.01887.
- [40] T. Chen, B. Xu, C. Zhang and C. Guestrin, Training deep nets with sublinear memory cost (2016); arXiv:1604.06174.
- [41] T. D. Le, H. Imai, Y. Negishi and K. Kawachiya, Automatic GPU memory management for large neural models in tensorflow, in *Proc. 2019 ACM SIGPLAN Int. Symp. Memory Management* (2019).
- [42] D. Oktay, N. McGreivy, J. Aduol, A. Beatson and R. P. Adams, Randomized automatic differentiation (2020); arXiv:2007.10412.
- [43] Y. Peng, Y. Zhu, Y. Chen, Y. Xin Bao, B. Yi, C. Lan, C. Wu and C. Guo, A generic communication scheduler for distributed DNN training acceleration. in *Proc. 27th ACM Symp. Operating Systems Principles* (2019).
- [44] J. Duarte *et al.*, FPGA-accelerated machine learning inference as a service for particle physics computing, *Comput. Softw. Big Sci.* **3**(1) (2019) 13; doi: 10.1007/s41781-019-0027-2.
- [45] J. Krupa *et al.*, GPU coprocessors as a service for deep learning inference in high energy physics (2020); arXiv:2007.10359.
- [46] C. Kim, H. Lee, M. Jeong, W. Baek, B. Yoon, I. Kim, S. Lim and S. Kim, torchpipe: On-the-fly pipeline parallelism for training giant models (2020); arXiv:2004.09910.
- [47] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez and I. Stoica, Tune: A research platform for distributed model selection and training (2018); arXiv:1807.05118.

Chapter 9

Machine Learning for Triggering and Data Acquisition

Philip Harris* and Nhan Tran[†]

**MIT 24-412, 77 Massachusetts Ave, Boston, MA 02139, USA
pcharris@mit.edu*

*[†]Fermilab, P.O. Box 500, Batavia, IL 60510, USA
ntran@fnal.gov*

With ever increasing data rates. The challenge of processing data at ever higher throughputs with more complex algorithms is daunting. However, new computing technology, and optimized algorithm design show a path towards resolving these large throughput issues. In this chapter, we present recent ideas on algorithm compression, quantization, and parallelization that allow for ultra low latency high throughput algorithms. Additionally, we present methods for longer latency algorithms that can be sped up through deep learning acceleration, leading to enhanced computational throughput.

1. Introduction

Many of the most significant advancements of deep learning have revolved around the development of specialized processors where parallelized computations can be performed natively and efficiently. This has been most directly evident from the prominence of graphics processing units (GPUs) for deep learning [1–3]. GPUs consist of many smaller cores that have a limited set of operations and slower clock frequencies than CPUs. Despite the limited available instruction sets, individual GPU cores can perform the typical addition and multiplication operations that enable large matrix multiplications. Additionally, GPUs have a large on-chip memory that allows for many iterative computations without the need to go off the processor

core. While GPUs have been shown to be excellent for the design, development, and training of deep learning algorithms, for real-time applications their functionality can be limited. Despite that, they represent a new avenue of specialized processor design that is quickly being populated by a number of different technologies. Following the first successful large scale deep learning implementations on GPUs, exploration of alternative processing technology using application-specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), and GPUs, have illustrated other ways to parallelize computations and allow for deep learning algorithms to be run ever faster and more efficiently. These advancements are enabling new, unforeseen, possibilities in neural network design. In this chapter, we investigate how these new technologies are opening up a new strategy to process data enabling the potential for deep learning within all tiers of data acquisition from real-time detector read out to detailed processing of the raw information.

In the late 2000s, the ability to distribute power to an ever-increasing number of transistors stalled, while the transistor density continued to increase. This ability had been driven by the fact that the required power to an individual section of the processor is driven by the local capacitance, which scales with the area. Consequently, as the size of a fixed number of transistors shrinks, so does the needed power, proportionately; thus, the transistor density can continue to grow. As transistors got progressively smaller, this scaling, known as Dennard scaling [4], started to break down because of the presence of leakage currents and threshold voltage [5]. Consequently, the frequency of a single processor became fixed, and the ability to perform processor operations was limited to a fixed processor size. This breakdown led to the rise of multi-core processing technology, which has dominated CPU design from 2010 to 2020. However, as multi-core processors become increasingly large, the limitations to power such a processor have become clear. In particular, for processors with 64 cores or more [6], the scaling of processing power per core starts to break down. In parallel to this approach, alternative processor designs such as GPUs have continued to improve, and, as of 2020, do not suffer from the same Dennard scaling limitations and can deliver a much larger number of computations per watt. With the rise of

artificial intelligence (AI) and machine learning (ML) techniques and the success of these processing technologies, specialized processors are starting to emerge as a critical element of future computing center design. Their high amount of parallelism, and different computational strategy have emerged as an approach to continually advance computation [7]. In low latency high throughput systems, such as those needed by high-energy physics (HEP) experiments, computational strategies are already unique. The addition of these new processing technologies, the advancement of deep learning algorithms compounded with the demands of high throughput have introduced a new, unexplored, computing challenge.

In addition to GPUs, other specialized processors are capable of highly parallelized computations. In particular FPGAs allow one to reconfigure the processor so that a specific set of instructions can be run in a highly parallelized optimal matter. Application-specific integrated circuits (ASICs) go a step further by explicitly restricting the set of instructions that can be programmed so that processing per watt can be further optimized. Various ASICs are emerging that target a broad range of processing strategies. These include tensor processing units (TPUs), intelligence processing units (IPUs), and the Cerebras processor, amongst many others.

Very recently, a new type of processor has emerged, the optical neuromorphic chip. In this scenario, laser light is sent through a processor, and optical waveguides with switching are used to perform multiplication and addition functions on the laser light. The time to complete the operations is roughly equal to the distance of the processor over the speed of light, or picosecond timescales. This well exceeds the abilities of existing electronic devices, and, as a consequence, limitations arise when coupling this to existing electronic devices [8].

1.1. *LHC dataflow*

In HEP, demands for low latency can arise in various scenarios and at varying timescales. To understand these demands, let us first consider the data flow process at the CERN large hadron collider (LHC) shown in Fig. 1. Dataflow at the LHC starts first with a detector

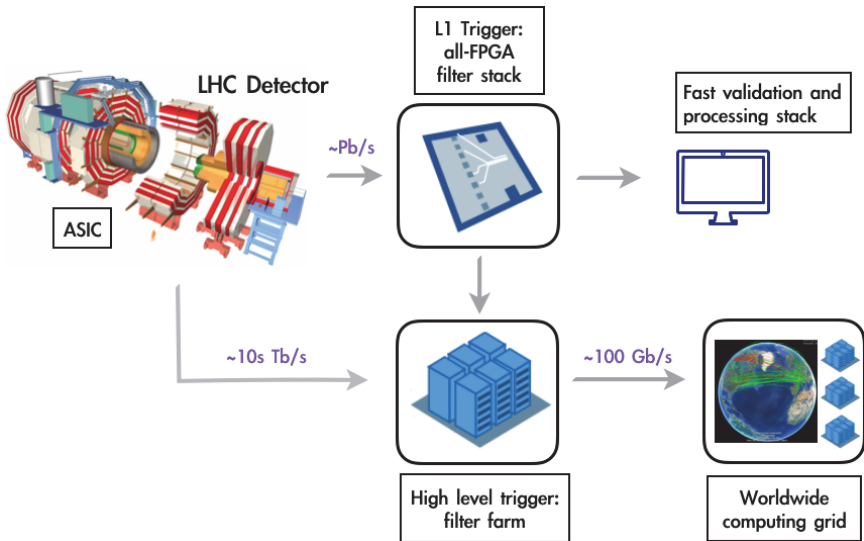


Fig. 1. The CMS processing chain as an example of the varied real-time processing challenges facing HEP experiments.

readout of a single collision. Collisions occur at a rate of 40 MHz. Consequently, the data readout of the detectors operates at a rate that is larger or at 40 MHz, a 25 ns interval between collisions. Due to the high radiation environment, data is readout with ASICs specifically equipped with additional redundancy to ensure robust readout in the high radiation collision environment. A compressed version of the data from the ASICs is then sent to a tiered set of processing boards consisting of FPGAs. This system performs reconstruction on a restricted set of detector elements and uses the resulting reconstructed output to determine whether it is sufficiently interesting to analyze further. The system is built of all FPGAs to allow for the large data rates to be transferred throughout the system. The total data rate of this system is approximately 50 Tb/s and will be upgraded to an amount approaching 1 Pb/s for the high-luminosity LHC (HL-LHC) upgrade. FPGAs with additional fiber links connected to the system can run at data rates of 25 Gb/s per fiber or 2.8 Tb/s between FPGAs. The high reconfigurability of FPGAs further allows the system to be progressively improved as algorithms

developed and collider configurations change. The first system at the LHC is responsible for reconstructing data at 40 MHz and identifying the most exciting events sent to later tiers of reconstruction. Within ATLAS and CMS, this tier is referred to as the L1 trigger [9–12]. At LHCb, this is referred to as the L0 trigger [13]. Event rates out of the first tier of reconstruction at CMS and ATLAS are reduced to a rate of 100 kHz. At LHCb, the output of the L0 trigger is used as a seed for full reconstruction. The output data rates from this first tier of reconstruction are on the order of several terabits per second.

Data rates on the order of a few terabits per second can be managed with a large computing cluster with optimized networking. Such data rates are comparable to the maximum capacity of other large scale computing systems [14, 15]. The second tier of LHC computing, called the high-level trigger (HLT) [16–18], like other computing systems, is constructed out of conventional computing hardware consisting of high speed networking, CPU cores, and, potentially, GPU cores [19]. Neutrino experiments, such as the deep underground neutrino experiment (DUNE), have data acquisition rates that are typically less than 100 kHz. These systems consist of a hardware based preprocessing system that goes straight to a computing cluster with comparable data rates and processing power to that of the second tier of LHC computing. The data rate that goes into this tier of computing is still quite large, and as a consequence, the data rate is reduced by another factor of one hundred. This is most often done by selecting one event in every hundred. However, “data scouting” or “selective persistence” approaches, whereby raw detector information is discarded and only reconstructed information is saved, have also been utilized to reduce the overall data throughput of this system [20–23].

The last stage of computing at the LHC and other big experiments, referred to as “offline”, processes data at a rate of roughly 1 kHz. This system consists of a large global distributed computing cluster. Despite many computing resources available, optimization in algorithm design is still a critical component of the reconstruction process at the LHC. As such, efficient use of deep learning algorithms is an essential element of the design of the system. The design and

implementation of deep learning systems in this scenario are similar to the design flow of the HLT.

Both at the L1 trigger, the HLT, and offline, Deep learning applications for LHC triggering and longer latency systems are quickly becoming a mainstay of detector design. Because deep learning is so highly parallelizable and relatively easy to develop, its application within online systems is natural, provided programmable tools exist to ensure the porting of algorithms to the appropriate high throughput system. The regular design of deep learning algorithms allows for flexible reprogramming, enabling the system to adapt to changing detector conditions by updating the weights. Despite these algorithms just starting to be developed within the triggering systems of the LHC, their adoption has been rapid.

At later stages of reconstruction, deep learning algorithms have been widely used [24–26]. However, there has been limited use on GPUs or other accelerators due to the relatively small size of the algorithms. With the development of large deep learning models within computer vision [27], natural language processing [28, 29], and physics [30–33], demand for large deep learning models will quickly grow. At that stage, the use of specialized processors becomes a necessity.

Within industry, focus on deep learning acceleration has centered on longer latencies. Applications within industry often operate at human timescales, and so they require overall latencies that are on the order of several milliseconds. For this reason, processor technology has usually focused on GPUs to process data. However, recently several use cases for faster deep learning algorithms have emerged. With Microsoft, the Brainwave system was constructed to deal with fast search for Microsoft Bing [34–36]. Bing search is driven by a tiered set of long short-term memory units (LSTMs) [37] that parse the search query through an iterative procedure of LSTM evaluations. Search algorithms are quite complicated and require a single query to be evaluated with very low latency; we refer to this as a batch size of one (batch-1) query. Low latency batch-1 queries need very fast inference calls that follow a paradigm that is quite different

from training a network. Optimized hardware, such as an ASIC or an FPGA, with a dedicated implementation of the network architecture is typically needed for this type of inference. For example, Bing search is currently FPGA-based. The use of FPGAs for Bing has been transformative and has elucidated an approach towards fast low batch inference engines. This approach is readily applicable to particle physics, where events are often processed on an event-by-event basis, which means large, event-level networks often require low batch. Dedicated ASIC and FPGA design for ML has also emerged for low power applications. This field, often referred to as “tiny-ML”, aims to develop optimized hardware for low power use, such that the AI algorithms can be run efficiently on remote devices where power is limited.

Given the varying data rates, and specialized architecture, the use and type of algorithms used within an L1 Trigger, and a HLT system are quite different. In Sec. 2, we present deep learning applications and design optimizations that are performed within the L1 Trigger. In Sec. 3, we present deep learning applications within the HLT. Finally, we summarize in Sec. 4 and present this work in the context of future developments.

2. Fast ML for Real-Time Readout and Near-Detector Triggering

Many HEP experiments produce data at extreme rates that cannot be stored and processed offline. In order to reduce the data volume to manageable sizes, real-time online data compression, zero suppression, and filtering is required. This online data processing begins at the sensor readout and continues all the way until data is stored for offline analysis. Many modern experiments require processing in custom hardware systems in addition to more traditional “edge” processing on local data center CPU-based servers. The deployment of ML algorithms in those custom electronics systems are the focus of this section while Sec. 3 will describe how to accelerate ML algorithms with heterogeneous computing paradigms.

2.1. *Hardware implementations of ML in real-time systems*

The online, real-time data processing chains for the highest rate HEP experiments require different types of hardware choices based on the latency, bandwidth, and environmental challenges of a given experiment. In Fig. 2, we give a *very* rough guide in latency for where off-the-shelf solutions are not sufficient (coprocessors) and custom hardware solutions are required such as FPGAs and ASICs. The most common system challenges in physics are low power constraints, cryogenic requirements, and high-radiation environments where specialized circuits (ASICs) are desired.

FPGAs can be conceptualized as reprogrammable circuits while ASICs are fixed circuits that are less flexible, but more optimized to specific tasks. For the purposes of deploying ML algorithms in such hardware systems, we can imagine these devices as resource-constrained non-von Neumann computing architectures that require a concurrent (as opposed to sequential) programming paradigm. Algorithm logic at its fundamental level is implemented in a hardware description language (HDL) that describes the behavior of electronic circuits. This HDL can be used to describe circuits in FPGAs or ASICs. The concurrent programming paradigm is very important for describing ML algorithms in hardware as the concurrency can be tuned to the needs of a given system to satisfy the resource and latency constraints as well as the system requirements such as power, area, and bandwidth.

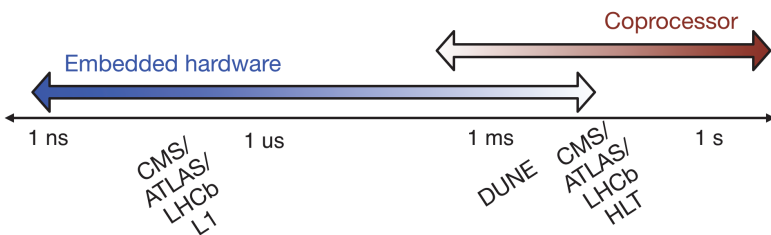


Fig. 2. The latency timeline for online processing.

Powerful AI instrumentation relies on **codesign** — the idea that system constraints, algorithm development, and hardware implementation inform and guide each other in complementary ways. One of the key features of neural networks (NNs) is their modularity. This allows development of programming paradigms that enable the developer to separate and recombine these specific modules to build larger NN architectures. The basic module-level description of the AI circuit implementation, for example, could be in a register-transfer level (RTL) language, but each module may be configurable based on resource, latency, and bandwidth constraints. One important recent development in hardware programming is the use of higher-level programming paradigms based on low-level hardware representations — the most popular of which is high-level synthesis (HLS) [38, 39].

2.2. *Parallel processing ML algorithms in constrained systems*

At a high level, FPGA and ASIC algorithm designs are distinct from programming a CPU in that independent operations may run fully in parallel, allowing FPGAs and ASICs to achieve trillions of operations per second at a relatively low power cost with respect to CPUs and GPUs. Local memory in registers or block random access memory units (BRAMs) allow fast memory access at very high speeds. However, such operations consume dedicated resources on the FPGA and cannot be dynamically remapped while running. The challenge in creating an optimal hardware implementation is to balance system resource usage against the latency and throughput goals of the target algorithm. Key metrics for a hardware implementation include:

- (1) **latency**, the total time (typically expressed in units of “clock cycles”) required for a single iteration of the algorithm to complete.
- (2) **initiation interval (II)**, the number of clock cycles required before the algorithm may accept a new input. The II is inversely proportional to the inference rate, or throughput; an II of two achieves half the throughput as an II of one. Consequently, data

can be pipelined into the algorithm at the rate of the initiation interval.

- (3) **resource usage**, expressed as a fraction of the available FPGA resources in each category: onboard FPGA memory (BRAM or URAM), digital signal processing (arithmetic) blocks (DSPs), and registers and programmable logic (flip-flops, or FFs, and lookup tables, or LUTs). For ASICs, in addition to the amount of logic, we also consider area and power consumed by the ML circuit and how it will fit into the rest of the chip design.

Pipelining is an important concept for achieving very high throughput for hardware systems. For example, at the LHC, inputs may be received every 25 ns (40 MHz) but the algorithm itself can take longer to run — from hundreds of nanoseconds up to microseconds. This is possible because the algorithm circuit is designed in the spirit of an *assembly line* of various modular tasks where new inputs can be received as soon as a given module is completed. This concept is very important as it is intrinsically tied to the parallelizability of an algorithm. Let us imagine a simple two-to-two dense layer as illustrated in Fig. 3. The right side shows that a given multiplier circuit can be used once, twice, or four times to perform the computation of this given dense layer. In the case of multiple uses of the multiplier units, new weights are sent to that multiplier unit from onboard FPGA memory. The more parallelized the algorithm becomes the more multiplier resources are required. However, if a single multiplier circuit is used to perform four multiplication for a given layer, then that layer’s II is four clock cycles — in other words, it cannot start the computation on new inputs for that layer until the previous computation is finished. The weight storage does not change with the reuse of multiplier units.

ML refers to the process by which we adjust the randomly initialized parameters of generic function approximators, termed “models”, so as to minimize an appropriately chosen loss function, or conversely to maximize a reward function. ML model architectures specify arrangements of “nodes”, usually co-evaluated in layers, where each node calculates the weighted sum of the inputs and a bias term,

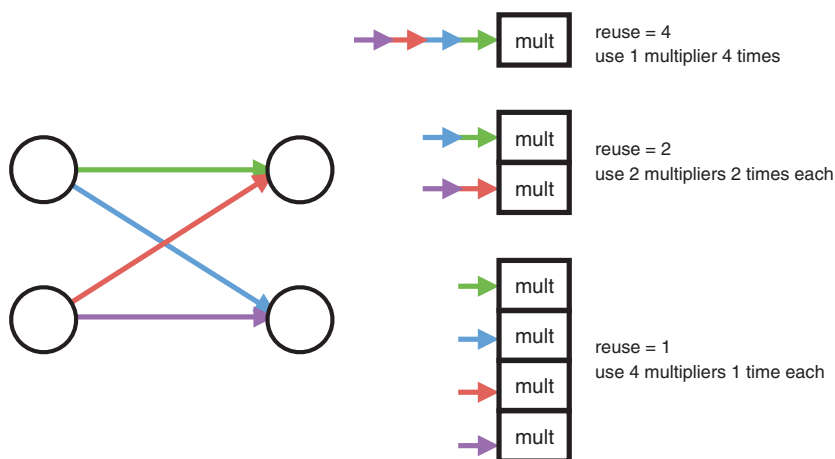


Fig. 3. Illustration of multiplier resource usage for different configurations of how often a multiplier circuit is used. The left drawing shows the case of two neuron pairs being connected by four connections, implying four multiplications to be performed. The right drawing shows how to perform these multiplications, from fully serial (top) to fully parallelized (bottom).

and outputs the value of a nonlinear “activation function”. The outputs of one layer typically provide the inputs to the next layer; in the simple multilayer perceptron (MLP) architecture, all nodes in each layer send copies of their output values to all nodes in the next layer. Layers not at the input or output are termed “hidden” layers. There are useful variations on this simple layer stack architecture such as recurrent NNs (RNNs), wherein some outputs from a previous forward inference are taken as inputs.

Most NN inferences can be characterized by the number of multiplication operations and parameters. Specifically the weighted sum computation within each layer translates to matrix multiply, which comprises a series of multiply-accumulate (MAC) steps on a processor. This can be translated conceptually into the number of allowed multipliers available in a given system, the latency and pipeline interval required, and the on-chip memory. The demands of the MAC step, and the desired parallelization, through II, and latency, provide basic constraints on how big or complex NNs can become for a given hardware system.

2.3. *Network design under hardware constraints*

With the knowledge of a given NN implementation into multiplications (operations) and number of parameters, we now have the inputs to map an ML algorithm into a rough hardware resource estimation. This is valuable in algorithm-hardware codesign as it helps us to iterate between the training process and algorithm performance and hardware optimization.

An important consideration in NN design is the type of architecture that is appropriate for your input data representation. For example, the type of architecture has implications for the hardware implementation. Once an architecture is chosen, hyperparameter tuning is no longer considering the network performance in terms of accuracy, AUC, etc., but also the latency and resource constraints of the system. However, hyperparameter optimization is not the only tool at one's disposal to create an efficient network in hardware.

NN inference can be made efficient with the following techniques: compression, quantization, and parallelization. We summarize these ideas briefly:

- **compression:** NN synapses and neurons can be redundant; compression attempts to reduce the number of synapses or neurons thereby effectively reducing the number of multipliers while maintaining performance;
- **quantization:** often 32-bit floating point calculations are not needed in the inference of a network to achieve optimal performance; quantization can reduce the precision of the calculations (weights, biases, etc.) in the NN with negligible loss in performance;
- **parallelization:** one can tune how much to parallelize the multiplications required for a given layer computation; in one extreme, all multiplications can be performed simultaneously using a maximal number of multipliers, while alternatively in the other extreme, one can use only one multiplier and perform the multiplications sequentially; between these extremes the user can optimize algorithm throughput vs. resource usage.

Considerable amount of work has been performed along all of these lines of development. Network compression [40, 41] is a widespread technique to reduce the size, energy consumption, and overtraining of deep NNs [42]. Several approaches have been successfully deployed to compress networks, including [40]:

- **parameter pruning:** selective removal of weights based on a particular ranking [42–44],
- **low-rank factorization:** using matrix/tensor decomposition to estimate informative parameters [45–49],
- **transferred/compact convolutional filters:** special structural convolutional filters to save parameters [50],
- **knowledge distillation:** training a compact network with distilled knowledge of a large network [51],
- **iterative unstructured pruning and retraining:** this uses L_1 regularization, where the loss function L is augmented with an additional penalty term, is known to produce sparse models, provide built-in feature selection [52], and is a readily available option in many ML workflows. The procedure is iterative and removes small weights from the network and is retrained,
- **lottery ticket methods:** these methods are similar to other types of parameter pruning, but involve “rewinding” weights and learning rates back to their initial conditions to discover sparse effective subnetworks present early in the training [53–55].

With quantization, algorithms started with post-training quantization, whereby the bit precision is reduced directly on the weights after the algorithm has been trained. Recently, a number of approaches to quantize algorithms have been reconsidered. Post-training quantization is a simple way to reduce the precision NN computations without loss in performance. In recent studies, the power of quantization-aware training (QAT) has been demonstrated [56]. With the prevalence and accessibility of new tools like QKeras [56, 57] and Brevitas [58, 59], QAT frameworks allow users to more easily train and deploy quantized networks. In certain cases, binary and ternary weights have shown very good performance as well. In Fig. 4, we show an example of the power of quantization at the training

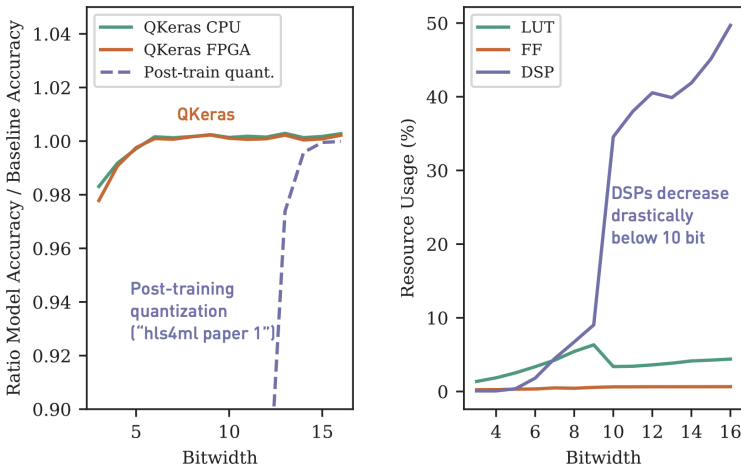


Fig. 4. Gains in perform through the application of quantization-aware training. The left plot shows the model accuracy compared to the ultimate accuracy as a function of bitwidth. The right plot shows the resource usage listed as look-up-tables (LUTs), logical flip-flops (FFs), and digital signal processors (DSPs) multiplier units [56, 57].

stage as compared to post-training quantization. The performance degrades at much lower bit precisions and the effect on multiplier resources is greatly reduced.

Lastly, parallelization of algorithm on hardware is a result of the constructed algorithm and its implementation. A broad range of possibilities exist from those that use dedicated matrix-multiply hardware blocks (systolic arrays) to approaches that define hardware specific algorithm design. In all cases, implementations are defined by the pipeline interval, and have largely followed a design flow whereby explicit resources are reused by their level of parallelizability (Fig. 3),

2.4. Examples of applications and outlook

In the previous sections, we described the basic concepts in the types of hardware for real-time data processing and filtering in particle physics experiments, algorithm/hardware codesign and resource constraints in hardware systems, and ML algorithm optimization

to enable efficient implementation in hardware. The convergence of these ideas in real hardware systems in science is only recently explored and we have just scratched the surface of real-world implementations and novel applications. The ability to detect, measure, and process sensor data at extreme rates and in extreme environments within complex systems has long been a unique necessity and strength of particle physics experiments.

We list here a number of tools for applications in particle physics:

- Boosted decision trees (BDTs) in the CMS L1 endcap muon trigger [60]: This is the first implementation of an ML algorithm in an LHC L1 Trigger. The algorithm used a large lookup table with external memory (1.2 GB) to combine 25 different variables in a BDT and demonstrated reduced rate of muon triggers in the CMS endcap muon trigger.
- NNs for signal processing in ATLAS tile calorimeter (TileCal) [61]: In this study, an NN is implemented to process pulse shape information in the TileCal. The performance is an improvement over classic algorithms, and the training and implementation is done using MATLAB tools.
- hls4ml [62]: This tool was introduced in [62] as a generic toolflow for translating NNs into FPGA implementations using high-level synthesis. The use of high-level programming paradigms enables fast algorithm/hardware codesign. It is demonstrated for jet substructure classifiers at the LHC. It has been used recently in many sub-microsecond L1 trigger applications [11] and has been extended with new features for binary/ternary networks [63], graph NNs (GNNs) [64–66], BDTs [67], and QAT [56]. The hls4ml toolflow has also been extended to ASICs for data compression and processing in high radiation environments [68].
- DL2HDL [69]: This tool translates models from deep learning frameworks into low level HDL and can be used for generic hardware codesign. It was used to employ RNNs to detect superconducting magnet anomalies (quenches) on the millisecond timescale. The algorithm uses MyHDL [70], a python-based HDL, for model translation.

The interest and potential of ML algorithms in real-time data processing is continuing to grow. There are many other potential use-cases which could continue to revolutionize experimentation in particle physics. While many of the first applications used traditional supervised classification and regression machine learning algorithms within reconstruction algorithms, there are many other potential use-cases. As deep learning techniques improve, NNs are beginning to replace entire algorithms. Additionally, unsupervised or weakly-supervised techniques [71–81] can be used to generically detect anomalies in the data, either for detector monitoring or searching for new physics. In most LHC applications, the ML algorithms are used for real-time filtering, but in other areas of physics, real-time intelligent processing can be used to provide feedback to the design and operation of experiments. Examples include the operation of detectors and control of accelerators. In detectors, real-time monitoring can be used to detect evolving experimental conditions and adapt algorithms to the changing environment. In accelerators, constant parameter tuning is vital for producing optimal beam conditions and those parameter adjustments can happen at timescales greater than 1 kHz [82–84]. These types of operational ML algorithms may require system-level design with both algorithm training and inference performed online. These algorithms are typically examples of reinforcement learning [85].

3. Fast ML for Data Processing

Many applications in HEP require deep learning algorithms that execute at timescales that are on the order of a few milliseconds or longer. These timescales are comparable to the demands of industry based systems, such as the application of a web-query algorithm or the processing of information for a self-driving car decision. The longer latency constraint of milliseconds, compared to the nanosecond timescales discussed in the previous section, allows for larger, more complex algorithms to be adopted. Additionally, it provides for the use of other processing techniques that do not give an immediate result for each sample, but can process a large amount of data

very efficiently; this is particularly true when considering GPUs. The use of GPUs and other alternative processing technology has remained limited within HEP. However, there is growing interest in developing tools to integrate GPUs and other parallelized processors into HEP computing workflows as heterogeneous coprocessors. In this context, deep learning may play a critical role in its design and implementation.

ML algorithms have been used in the trigger systems of colliders since the 1980s [86–92]. A wide range of algorithms are already present within existing LHC systems and are becoming increasingly popular. GPUs and other processing techniques have opened the door to potential redesigns of these systems that can lead to larger throughput systems and significantly more effective designs. The demands of millisecond latency neural-network inference are sufficiently slow that conventional tools within industry can be adapted for use within high energy physics.

3.1. Previous use of ML in HLT systems

Typical input event rates for an LHC HLT system are roughly 100 kHz. At modern neutrino detectors, input event rates are similar, ranging up to the several tens of kilohertz. At the Tevatron, a deep NN identifier was used to select hadronic decays of tau leptons [93]. At the LHC, within the HLT, many ML approaches have been used.

Early, large scale use of ML within the HLT was performed within the LHCb collaboration to identify bottom quarks [94, 95]. In this algorithm, a regularized BDT was utilized (Bonsai BDT). For the trigger, the ideal strategy is to keep the algorithm computationally efficient and robust against changes in detector conditions. Since BDTs consist of a weighted set of binary selections to classify the event, the strategy to build an efficient algorithm was to minimize the number of possible branchings and the number of input variables. The final algorithm was constructed from seven high-level features and iteratively trained with a progressively coarser discretization of the input variables to limit the number of possible options and keep the computation of the BDT fast.

Both ATLAS and CMS have deployed ML algorithms for various aspects of the reconstruction in the trigger system. For calorimeter reconstruction, ATLAS developed a one-hidden-layer MLP that considers energy deposits from concentric calorimetric rings [96, 97]. The number of inputs was optimized to ensure fast CPU-based inference time, and concentric square rings ensured fast computation. The full algorithm with feature computation and inference takes 1.1 ms for all 17,000 regions of interest. This algorithm led to a reduction in background rates by a factor of two and was used as a seed for electron reconstruction to reduce the number of seeds for dedicated track reconstruction. This approach avoids computations of more complex features instead using more easily computed ring-based variables. More recent developments along these lines consist of sending raw detector information directly to an ML algorithm, and letting the algorithm “construct” features through the network architecture [98]. In both ATLAS and CMS, higher level physics objects within the HLT also extensively use ML, including hadronic τ lepton identification in ATLAS [99], bottom quark tagging in CMS [24], and electron identification in CMS [100].

The prevalence of ML algorithm development has mostly paralleled the advancement of GPUs. GPUs and other processor technologies can perform highly parallelized computations allowing for the possibility of many calculations performed in parallel. While there are limitations for the use of GPUs in the L1 trigger because of restrictions in their data ingestion rates, and restricted design flexibility, these limitations are not present for the HLT. Thus, they have been considered in several instances over the past 10 years.

Although no deep learning algorithm has been deployed on a GPU within a triggering system in HEP, GPUs have been used for traditional algorithms. GPUs were first integrated into the 180 compute nodes of the HLT workflow of the ALICE experiment at the LHC to perform charged particle reconstruction. They were part of the ALICE operations during 2010–2013 and 2015–2018 [101]. More recently, GPUs are being considered for the HLT of the LHCb experiment [102, 103]. This HLT system relies entirely on charged particle tracking algorithms. Additionally, within the CMS experiment,

GPUs have been explored for charged particle reconstruction through the use of cellular automata [104], an accelerated pattern recognition algorithm [105], and ported conventional tracking and vertexing algorithms [19]. Additionally, ATLAS deployed a GPU demonstrator that could perform inner tracker reconstruction, and topocluster reconstruction at a rate slightly worse than a CPU at the time [106]. To date, neither CMS nor ATLAS have used GPUs in HLT operation. Beyond charged particle tracking, algorithms for GPUs and FPGAs have been developed (but not implemented) for real-time processing of ring imaging Cherenkov detectors for the LHCb HLT [107, 108]. Beyond the LHC experiments, GPU algorithms have been developed for the trigger readout of the Mu3e experiment [109], and the dark matter experiment NA62 [110]. These algorithms are planned to run in the next round of data taking for each experiment, starting in 2021. In all instances, the GPUs have been directly connected to the CPUs via PCI Express (PCIe).

3.2. Millisecond latencies with deep learning algorithms

The scope of processors is shown in Fig. 5. Processors are divided into four main categories of processor **CPU**: a processor that can perform single complex operations at a high rate, however with parallelism limited to vectorized operations, **GPU**: a processor capable

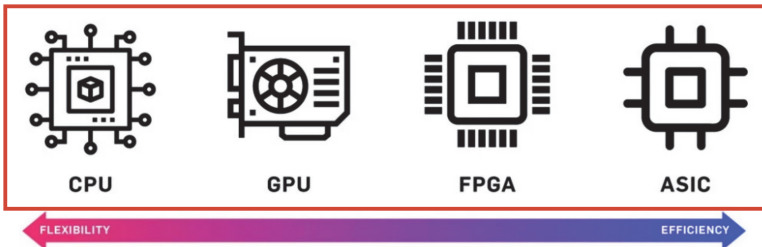


Fig. 5. Range of processor technologies for deep learning applications. On the left side, we have conventional processors, which are capable of performing a large range of computations quickly on the fly. On the right side, we have FPGAs, and ASICs, which require a significant amount of programming.

of running many simplified operations in parallel, **FPGA**: a processor with individual multiplier and logic units that can run in parallel and that can be reprogrammed to perform a specific task, and **ASIC**: a processor that is hardwired to perform a particular task, and has a limited ability to be reprogrammed. Ranging from CPUs to ASICs, the processor technology is increasingly more parallel and optimized for specific operations. However, this leads to reduced flexibility in the scope of operations and reprogramming that the processor can undertake. As a consequence, specialized processors have had limited adoption. Deep learning has altered this landscape because of the enormous potential for parallelism. Very recently, a new class of ASIC processor has emerged that is explicitly designed for large-scale computation of neural network operations often through systolic arrays. This class of processors includes the newly developed tensor processing unit (TPU) and the Intel Habana processor among others.

Alternative processors like GPUs, FPGAs, and ASICs have the ability to run significantly larger algorithms at the HLT. These algorithms can avoid significant computation from the determination of high-level features to be fed into the algorithm. In place of these high-level features, raw detector inputs can be fed directly into the network, and the network can perform the bulk of the processing. The use of raw inputs potentially allows for reduced overall complexity at the expense of expert engineered features within the algorithm. To understand where gains are possible in this approach, we consider two fundamental features: processing power and throughput.

Processing power dictates how the algorithms should be distributed. Extensive algorithms, with a large number of weights that require many operations, are best used on processors where there is a large available memory so that weights and intermediate computations can be stored on the processor as it works through the computation. The throughput on these processors tends to reflect the complexity of the algorithm, and they are often not designed for low latency computation. Smaller processors with less on-chip memory are often better for low latency and, sometimes, low power

computation, assuming a relatively small number of computations on a single processor.

3.3. *Processing power*

NN computations can be reduced to multiplications and additions, often referred to as multiply-accumulate operations (MACs). The number of MACs depends on the network size. For dense networks, this equals the total number of weights in the model. For RNNs, this is equal to the number of weights multiplied by the number of recursions. For convolutional NNs (CNNs), GNNs, and other networks that rely on regionized calculations, calculating the number of operations is more complicated and can be considerably larger than the number of weights. The approximate throughput of any NN algorithm on a processor can be determined by considering the number of operations per second (OPS). Large processors such as TPUs and large GPUs can currently run at about 100 TOPS [111]. CPUs, on the other hand, can achieve roughly 12 computations per core per clock cycle at 4 GHz for a total of 48 GOPS per core [111]. This means that a single GPU can perform the same number of computations as approximately 5000 CPU cores, albeit with a limited instruction set. Large CPUs consist of as many as 64 or 128 cores, which leads to an imbalance in computing power per processor of nearly two orders of magnitude. Such an asymmetry means that when designing algorithms to be used on heterogeneous processing systems, they either have to serve many compute nodes, or they need to comprise a large portion of the total amount of computing required.

A second important consideration is that a large number of operations usually implies a large number of computational results are produced. To store these results efficiently, processors utilize memory. Significant on-chip memory helps store intermediate computational steps so that large computations, such as NNs with millions of weights, can be performed quickly and over a large set of input objects. Memory directly located on the processor can be accessed on nanosecond timescales. Consequently, the full throughput available

internally on the processor can be exploited to yield the computational result.

Power utilization of the processor is driven by the number of operations and the total amount of memory used. GPUs, and ASICs designed for large batching tend to use a large amount of power to contend with the many operations, and large number of memory transfers. FPGAs, and specialized ASICs use considerably less power due to their ability to be programmed efficiently for the specific network architecture desired. CPUs tend to be worse than all others. Power rates are quoted in terms of TOPs/watt with a typical processor achieving 1 TOPs/watt [112].

3.4. *Throughput*

Within computing clusters, GPUs, FPGAs, and ASICs are typically connected to other processing elements through a PCIe connection. Communication through the PCIe connection can be a bottleneck with a limit of roughly 10–64 Gb/s [113]. Additionally, the transfers are limited by the packet size and compression. Consequently, additional time is lost because of preprocessing and data transfer. The time lost translates to 0.1–0.5 ms; this bound limits the use of coprocessors to algorithms that take more than 1 ms for a single batch. Batch processing consists of performing the same NN inference on a “batch” of inputs. Processors, such as GPUs, are designed for large batched operations. While they are relatively slow when considering the inference of a single input, they can perform many inferences in parallel leading to high throughput under large batches. There have been a variety of ideas to avoid the bottleneck of PCIe communication [114]. Most of these involve direct communication to the processor through a network connection that goes straight to the coprocessor [113, 115]; this is currently being done through several protocols including NVLink, UDP, TCP/IP, and PCIe.

3.5. *Algorithm optimization for processors*

When designing algorithms for high throughput on specific processors, several considerations need to be taken into account. The most

critical elements are the network size and the typical batch size. To illustrate the importance, we consider two implementations of the industry standard ResNet-50 algorithm [27], which has also been adapted to perform top quark identification [116].

ResNet-50 is an image recognition algorithm capable of outperforming humans. It consists of 50 CNN layers and classifies images into 1000 different categories. Several FPGA implementations exist of ResNet-50. Each of these implementations processes a single image at a time (batch-1 processing); the results shown in Fig. 6 include the ResNet-50 FPGA implementation available on Microsoft Azure in which a single image can be processed within 1.5 ms. To ensure optimized performance, the FPGA exploits layers with dynamic bit precision to minimize the computation without degrading the overall image recognition. For comparison, batch-1 inference on an equivalent GPU takes 7 ms. However, when batch size of 50 is utilized, the throughput on the GPU is the same as the FPGA, despite a long overall latency for any particular image (50×1.5 ms). Lastly, if this were to be performed on a CPU, the batch-1 processing time

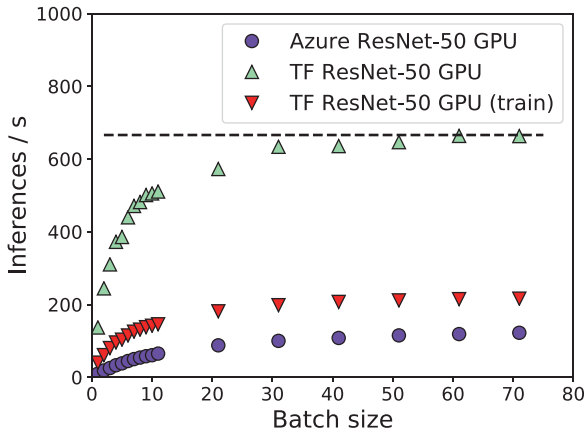


Fig. 6. (Black dashed line) Comparison of an FPGA implementation (Artix) of ResNet-50 (batch-1) with a GPU (Nvidia 1080Ti) implementation of ResNet-50 where the batch size is varied (TF ResNet-50). The exact FPGA-based implementation (Azure ResNet-50) is also run on a GPU for comparison. The training labeled algorithms shows the processing speed on the same GPU when algorithm training is performed in place of inference.

would be approximately 500 ms for a single CPU core. The much larger latency compared to either an FPGA or GPU demonstrates the significant advantages possible with a coprocessor.

GPU performance suffers as the batch is decreased because the parallelized computation is underutilized. The full use of the GPU is only realized with a large batch. The FPGA, on the other hand, aims at the optimized performance of each layer of the network. Thus, the full processor is utilized in performing the network inference even for a batch size of 1. More recent ResNet-50 implementations on GPUs and FPGAs exist that have larger throughput. However, this illustrates the impact of processor design on how it impacts model acceleration.

3.6. Inference as a service with heterogeneous computing

Accessing the high computational capacity of alternative processors connected to a host CPU, known as coprocessors, requires careful consideration of the communication between the CPU and the coprocessor in order to optimize the processing of the data. To enable the use of multiple processors, several options have been investigated:

- Communication with a coprocessor through a direct connection (PCIe slot).
- Communication with a coprocessor server through the network to a CPU serving coprocessors.
- Direct communication with a coprocessor over a network.
- Coprocessor to coprocessor communication.

In the first approach, a single CPU directly sends instructions to the coprocessor connected through a PCIe slot. In the second approach, a CPU connected to one or multiple coprocessors acts as a server and takes instructions from many other CPUs through network calls. This approach is called “as-a-service” computing. In the third approach, the CPU intermediary is eliminated, and the processor communicates directly with other CPUs over the network. This approach is referred to as “bump-in-the-wire” and has had successful, but limited use.

The final approach is only used in dedicated systems, such as for the L1 trigger discussed earlier in this section. While bump-in-the-wire technology is quickly gaining headway for industry applications. The only documented use case of direct network to coprocessor connection for high-energy physics was performed on the FPGAs in the Azure ML cluster using ResNet-50. Given the limited use of bump-in-the-wire, and direct processor to processor technology, we consider the first two approaches in the context of future accelerated computing models.

Deploying coprocessors (FPGA, GPUs, or ASICs) as a service is a widely utilized approach to incorporate alternative coprocessors, and we find that it has several advantages over a direct-connect approach. Deploying coprocessors as a service:

- (1) increases hardware cost-effectiveness by reducing the number of coprocessors required to achieve the same throughput. This is possible since each coprocessor can service many more CPUs than a direct-connect paradigm would allow,
- (2) augments our existing computing model only through offloading the specific algorithms with large speedups with minimal client-side re-configuration,
- (3) facilitates easy integration and scalability of heterogeneous coprocessors (such as GPUs and FPGAs), as suited for optimal algorithmic performance, and
- (4) exploits existing open source frameworks that have been optimized for fast coprocessor inference and are widely used.

These advantages help to balance the resource load, but they come at the cost of added networking, and the requirement to have optimized schedulers.

To provide deep learning inference as a service, an algorithm is chosen with a significant speedup when using a coprocessor. Existing server toolkits exist to perform the integration of GPU, and FPGAs as a service [62, 117–121]. These tools are capable of load balancing both within a single coprocessor and among many coprocessors. Calls can be made to these servers using standard network protocols.

The most common currently used protocol is the Google Remote Procedural Call (gRPC) [122]. This protocol builds on a TCP/IP back end to deliver optimized high throughput delivery to the server. A significant amount of work both in industry and within high-energy physics is underway to harmonize the protocols so that standardized integration of coprocessor servers can be done efficiently with minimal overhead [118].

A modified reconstruction workflow depicted in Fig. 7 shows how deep learning inference as a service can be deployed for parallel tasks without limiting the overall throughput of the system. Since the coprocessor is running disjoint from the actual reconstruction workflow, both can be run in parallel with some time lost to package the request to the coprocessor, and, potentially, time lost waiting for the final results of the coprocessor.

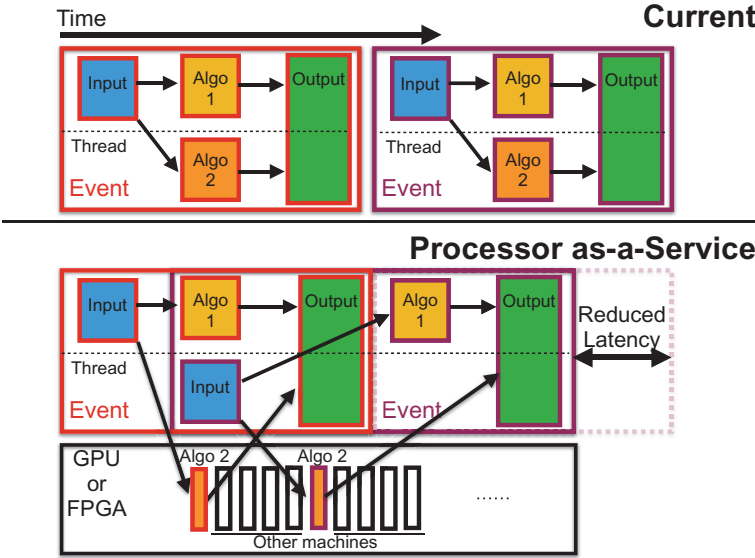


Fig. 7. Diagram comparing the traditional LHC production model on CPU with the GPU or FPGA as-a-service approach. Each block represents a module within the reconstruction framework. For the GPU or FPGA as-a-service approach, algorithm 2 is run on the GPU or FPGA, which allows the processing of the second event (outlined in purple) to run concurrently with the first event (outlined in red) [118].

Software frameworks in HEP have parallelized much of the scheduling of reconstruction workflow through task-based multi-threading available on multi-core processor frameworks [123]. This facilitates asynchronous, non-blocking calls to external resources [124], which is the most efficient way to utilize coprocessors as a service because of the CPU running the experiment software can do other work while the service call finishes.

A final additional feature present through the as-a-service paradigm is the possibility of dynamic batching [125]. Dynamic batching, shown in Fig. 8, is a feature that serves to increase both the throughput and hardware efficiency. Dynamic batching creates a server-side queue of requests from the many clients being served and continuously aggregates events until an optimal batch size is reached. The aggregation is limited to within a chosen time window (e.g. less than $200\,\mu\text{s}$). This yields an optimization problem between the dynamic batch size and model concurrency. The use of dynamic

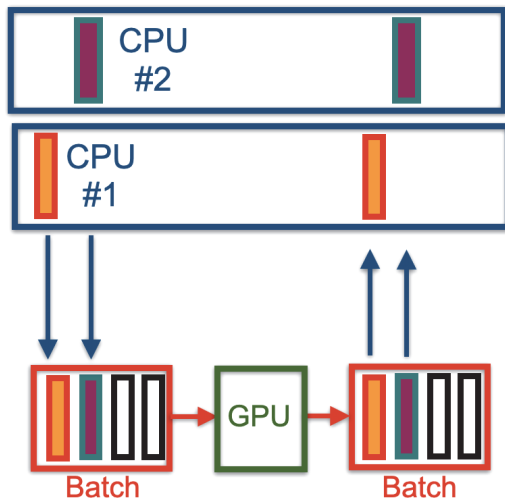


Fig. 8. Diagram illustrating how dynamic batching can be used to send optimized calls to the GPU by allocating calls to the server over a specified time window. Signals from many CPUs are aggregated into a single batch that limits the calls to the GPUs. The aggregation is performed within a chosen time window so as not to delay the overall computation.

batching is particularly interesting because it circumvents the HEP computing paradigm of splitting computations on an event-by-event basis. Here, multiple events can be processed simultaneously within a single computation, without redesigning the computing model. We stress that this type of scheduling is *only* beneficial when GPUs are servicing a large number of parallel processes.

3.7. Metrics for optimization

To describe important elements of the as-a-service computing model, we first define some concepts and variables.

- t_{CPU} is the total time for CPU-only inference.
- p is the fraction of t_{CPU} that can be accelerated, and conversely $1 - p$ is the fraction of the processing that is not being accelerated.
- t_{XPU} is the time explicitly spent doing inference on a generic heterogeneous processor XPU.
- $t_{\text{preprocess}}$ is the time spent on the CPU for preprocessing to prepare the input data to be transmitted to the server in the correct format.
- t_{transmit} is the latency incurred from transmitting the neural network input data.
- t_{travel} is the travel latency to go from the CPU to the XPU server.
- $t_{\text{latency}} = t_{\text{preprocess}} + t_{\text{transmit}} + t_{\text{travel}}$ summarizes the additional latency distinct from the actual XPU processing.
- t_{ideal} is the total processing time assuming the XPU is always available.
- N_{CPU} and N_{XPU} are the numbers of simultaneously running CPU and XPU processors, respectively.

With each element of the system latency now defined, we can model the performance of utilizing as-a-service computing. Initially, we assume blocking modules and zero communication latency. The total time of a CPU-only job can be written as

$$t_{\text{CPU}} = (1 - p)t_{\text{CPU}} + pt_{\text{CPU}}. \quad (1)$$

We replace the time for the accelerated module with the XPU latency terms

$$t_{\text{ideal}} = (1 - p)t_{\text{CPU}} + t_{\text{XPU}} + t_{\text{latency}}. \quad (2)$$

This reflects the ideal scenario when the XPU is always available for the CPU job. We also include t_{latency} , which accounts for the preprocessing, bandwidth, and travel time to the XPU. The value of t_{XPU} is fixed, unless the XPU is saturated with requests. The average number of requests a given XPU is receiving is $N_{\text{CPU}}/N_{\text{XPU}}$. The maximum number of requests a given XPU can receive, which does not delay the process is $t_{\text{ideal}}/t_{\text{XPU}}^{\text{ideal}}$. Above this the cumulative time spend per XPU exceeds the total time. The saturation condition is therefore defined as

$$\frac{N_{\text{CPU}}}{N_{\text{XPU}}} > \frac{t_{\text{ideal}}}{t_{\text{XPU}}^{\text{ideal}}}. \quad (3)$$

Here, t_{ideal} is defined in Eq. (2), and $t_{\text{XPU}}^{\text{ideal}}$ are the respective processing times assuming there is no saturated XPU. When the accelerated process is not saturated, there is no increase in the overall latency of the system since the processor is always available. However, once the processor is saturated with calls from many parallel processes, the latency is then driven by the number of parallel processes. To explain this, we define

$$N_{\text{excess proc}} = \frac{N_{\text{CPU}}}{N_{\text{XPU}}} - \frac{t_{\text{ideal}}}{t_{\text{XPU}}^{\text{ideal}}}, \quad (4)$$

which represents the number of addition CPU cores relative to XPU cores above the saturation condition.

Consequently, there are two conditions, unsaturated and saturated XPU, which correspond to $N_{\text{CPU}}/N_{\text{XPU}} < t_{\text{ideal}}/t_{\text{XPU}}$ and $N_{\text{CPU}}/N_{\text{XPU}} > t_{\text{ideal}}/t_{\text{XPU}}$, respectively. The time taken on the XPU when the system is saturated $t_{\text{XPU}} \neq t_{\text{XPU}}^{\text{ideal}}$ is thus written by the saturation condition on the XPU, which we can write as

$$t_{\text{XPU}}^{\text{saturated}} = t_{\text{XPU}}^{\text{ideal}} \frac{N_{\text{CPU}}}{N_{\text{XPU}}} \quad (5)$$

$$= t_{\text{XPU}}^{\text{ideal}} \left(\frac{t_{\text{ideal}}}{t_{\text{XPU}}^{\text{ideal}}} + N_{\text{excess proc}} \right) \quad (6)$$

$$t_{\text{XPU}}^{\text{unsaturated}} = t_{\text{XPU}}^{\text{ideal}}. \quad (7)$$

Combining the saturation condition and the unsaturated condition, we can compute the total latency (t_{total}) to account for both

cases:

$$\begin{aligned}
 t_{\text{total}} &= (1 - p)t_{\text{CPU}} + t_{\text{XPU}}^{\text{ideal}} [1 + \max(0, N_{\text{excess proc}})] + t_{\text{latency}} \quad (8) \\
 &= (1 - p)t_{\text{CPU}} + t_{\text{XPU}}^{\text{ideal}} \left[1 + \max \left(0, \frac{N_{\text{CPU}}}{N_{\text{XPU}}} - \frac{t_{\text{ideal}}}{t_{\text{XPU}}^{\text{ideal}}} \right) \right] \\
 &\quad + t_{\text{latency}}. \quad (9)
 \end{aligned}$$

Therefore, the total latency is constant when the processors are not saturated and increases linearly in the saturated case proportional to $t_{\text{XPU}}^{\text{ideal}}$. Substituting Eq. (2) for t_{ideal} , the saturated case asymptotes, in the large N_{CPU} limit, to:

$$t_{\text{total}} = t_{\text{XPU}}^{\text{ideal}} \frac{N_{\text{CPU}}}{N_{\text{XPU}}}. \quad (10)$$

Consequently, for an optimized system, we can consider an optimal use of the coprocessor when we balance the right amount of processors to reflect the relative latency of each algorithm. This ratio further defines a rate at which an XPU can replace an alternative processor that can be used when designing an extensive system. Finally, when the use of the coprocessor is fully asynchronous and non-blocking, we find that impact of the throughput from the coprocessor can be removed since the asynchronous scheduler can run reconstruction of other components that do not require the coprocessor. As a consequence, we obtain an average latency per process of

$$t_{\text{ideal-async}} = (1 - p)t_{\text{CPU}} + t_{\text{preprocess}}. \quad (11)$$

This is only true provided the saturation condition is not met and the system is not completely saturated. Finally, for a 32-core CPU attached to an XPU through PCIe, we find that it is better to use as-a-service computing when $t_{\text{total}}/t_{\text{XPU}} > 32$.

3.8. Applications

Deep learning inference for sub-second timescales has a broad range of applications in physics. As with the LHC, where nanosecond timescales are sometimes needed, the timescales are largely dictated by the detector technology and the underlying physics process. Given

the availability of many different processor technologies, the latency and throughput demands are crucial to optimizing the deep learning throughput with the existing processor technology. Below, we provide an incomplete list of various applications within physics that demand low latency deep learning inference.

LHC HLT: The HLTs of the ATLAS, CMS, and LHCb experiments run at a rate of 100 kHz on a cluster consisting of several 10,000s of cores. This equates to an individual event processing time that is on the order of 0.5 s, lending itself well to the possibility of running deep learning inferencing with all possible coprocessors provided results are available on the 10–100 ms timescale.

Triggering neutrino events from supernovae: To monitor the possibility of Supernovae events, neutrino experiments aim to continuously readout and process data at a rate of 100 kHz, driven by the readout time of typical neutrino detectors. This rate and the event size are of similar scale to the LHC high level triggers and would benefit from deep learning inference on a similar timescale.

Transient event identification: Astrophysical transient events can occur on timescales less than 10 ms. Processing these events involves a significant amount of data, which need to be processed in near real time to allow for the possibility of cross correlation with other experiments capable of observing similar astrophysical phenomena.

Gravitational wave reconstruction: Like other astrophysical transients, gravitational wave signals occur on timescales ranging from 50 ms to several minutes. Fast, near real-time reconstruction of the event waveforms is essential to ensure the possibility of cross correlation of events with other experiments. The field of cross-correlation is known as multi-messenger astronomy.

3.9. *ProtoDUNE and LHC applications*

As an illustration of the effectiveness of deep learning based acceleration as a means to speed up real-time processing of events, we consider two examples. These two examples, paired with the previously presented results with ResNet-50 in Fig. 6, are intended to

illustrate the effectiveness of deep learning on coprocessors at reducing the overall reconstruction time and increasing the throughput. The goal of this section is to be illustrative as it is likely that more sophisticated algorithms will supersede these algorithms and many other newer algorithms will be introduced.

With the LHC, we consider an ML algorithm that replaces the current hadron calorimeter (HCAL) reconstruction of a single channel within the high level trigger. Within ProtoDUNE, we consider an algorithm that performs Michel electron identification on a small patch of the detector. Both algorithms need to be applied on many elements of their respective detectors when analyzing a single event. The LHC algorithm is a very small dense NN consisting of a few thousand weights, whereas the ProtoDUNE algorithm is a moderately sized CNN.

3.9.1. *Hadron calorimeter reconstruction (FACILE)*

A benchmark deep learning example is the fast calorimeter learning (FACILE) algorithm, a small deep neural network composed of 2000 parameters and five fully-connected NN layers. This algorithm was trained on simulated collisions to reconstruct the energy deposited by particles in the HCAL subdetector of the CMS experiment at the LHC. Particles that interact in the HCAL consist of any particle with quarks (a hadron); this is the large majority of all particles in the collision. We can expect as many as 1000 separate hadrons in an LHC collision. The HCAL is a core component of LHC experiments and a prototypical subdetector for implementing ML as-a-service reconstruction. FACILE is run separately on all 16,000 HCAL channels in CMS or, in other words, with a batch size of 16,000.

To ensure robust performance, we find that local and global objects reconstructed with FACILE have as good or better resolution compared to the nominal algorithm that does not use machine learning. Second, the nominal HCAL reconstruction algorithm at CMS consumes about 60 ms of CPU time in online reconstruction, accounting for approximately 15% of the online computing budget [126]. FACILE offers a significant improvement in computing performance

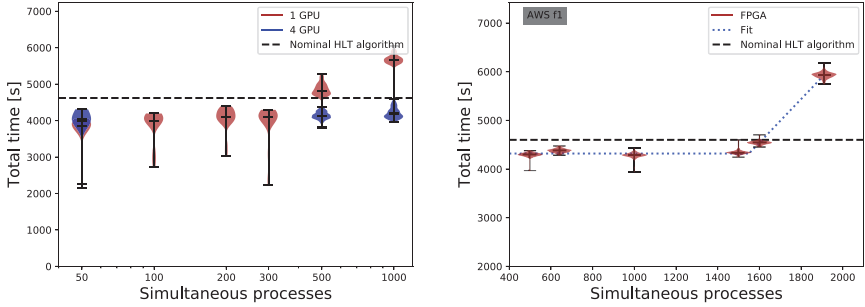


Fig. 9. (Left) Performance of the total HLT time for a 1 or 4 GPU server with the FACILE algorithm deployed as a function of the number of simultaneous processes being run. (Right) Performance of the total HLT time as a function of number of simultaneous HLT processes being run when FACILE is deployed on an FPGA server with a 25 Gb/s connection. The saturation point occurs at the maximum network bandwidth and not at the ultimate limit from the processor [118].

when operated as a service by reducing the CPU time to less than 7 ms and the GPU throughput to 2 ms, resulting in a GPU replacement ratio of $t_{\text{CPU}}/t_{\text{GPU}} = 27$. For an FPGA, the throughput time per 16,000 channel inference is found to be 100 μs or an FPGA replacement ratio of $t_{\text{CPU}}/t_{\text{FPGA}} = 540$.

Figure 9 shows the performance of FACILE integrated into the CMS HLT compared to before [127–129]. A drop in the per-event latency is observed of more than 10%. The drop is consistent with the maximum reduction expected given the observed speed-up of the algorithm on either an FPGA or GPU. When running the algorithm with a GPU, we find a single GPU can serve more than 300 individual HLT processes. With an FPGA this number saturates at nearly 1600 parallel HLT processes. The saturation at 1600 is a direct result of Networking limitations of the FPGA server. On site tests show that the full FPGA number would potentially be able to serve 3000–4000 separate HLT nodes, or in other words 10 FPGAs would be able to reduce a 40,000 core system by more than 10%.

FACILE is particularly well suited for an FPGA, the simple design and small size of the network make it easy to implement on an FPGA. To convert this algorithm, we utilize our FPGA compiler hls4ml. The output of hls4ml in this case is an FPGA kernel that accepts

all inputs simultaneously and produces the output in 17 clock cycles, with a target clock frequency of 250 MHz. This means that the inference result is available in 68 ns.

3.9.2. *ProtoDUNE reconstruction algorithm*

Neutrino experiments typically consist of a large volume detector that allows for the possibility of neutrinos to interact anywhere within the volume. Neutrino interactions in the volume consist of a series of energy deposits in tracks emanating from a single source. The algorithm used in this study is the identification of Michel electrons [130] with the ProtoDUNE detector. Michel electrons are electrons from muon decays. The low energy signature occurs at similar energy to neutrino events from supernovae. However, the signature is distinct from neutrinos and thus they are an excellent calibration tool. The ProtoDUNE algorithm consists of a CNN of about 11 million trainable parameters. The CNN is applied to 48×48 pixel sub-images within the ProtoDUNE detector. There are a total of 55,000 subimages in the whole detector. Processing of the data is performed in small or large batches of 235 or 1693 depending on the detector geometry.

Due to the large number of sub-images, and the complexity of the model, when run on a CPU, this algorithm takes approximately 2/3 (220 of 330 seconds per event) of the total ProtoDUNE reconstruction time. Originally, this algorithm was run on a CPU. Now, due to the integration of GPU-as-a-service into liquid argon time projection chamber (LArTPC) reconstruction software, it is now possible to offload the Michel electron identification algorithm onto a GPU [120]. Figure 10 shows the performance of the algorithm as a function of the separate compute nodes interacting with the four-GPU server. When run on a CPU, the algorithm takes 220 s, while on a GPU the total time is about 7 s. The actual time on the GPU is found to 1.9 s or a speed of more than a factor of 100. Additionally, it is found that four GPUs can server nearly 280 CPUs running ProtoDUNE reconstruction.

The ProtoDUNE algorithm is an example where GPUs are expected to perform best. The relatively large complexity of the CNN

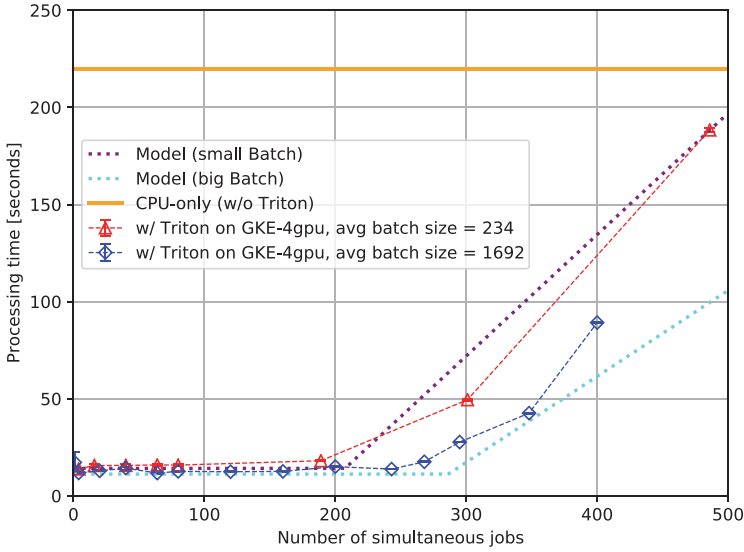


Fig. 10. Processing time for the Michel electron identification algorithm as a function of simultaneous CPU processes, using a 4-GPU server. The orange line shows the nominal algorithm latency. In both plots, the dotted lines indicate the predictions of the latency model, specifically Eq. (9) [120]

algorithm, the ability to use large batch sizes, and, in this case, the limited demand for low latency makes it conducive to use GPUs for the processing of this operation, an FPGA implementation, if it existed, would likely perform better at low batch, but worse at high batch. Lower batch size or a significantly smaller model size would have led to different conclusions. Lastly, in Fig.10, a comparison is placed against the model presented in Eq. (9). Direct comparison shows similar trends and indicates that the guiding principles of the heuristic model are confirmed by observation.

The examples in ProtoDUNE and CMS represent two examples of deep learning inference acceleration as a test of the reconstruction workflows in neutrino and collider physics, respectively. Future developments have the potential to explore larger models, or further optimized throughput. The choice of these examples is to illustrate where clear algorithmic speedups appear as a result of the parallelism present in ML algorithms.

4. Concluding Remarks

In this chapter, we have presented fast machine learning for real-time readout of LHC collisions at sub-microsecond fixed latency, and in the context of high-throughput offline processing and millisecond-level real-time systems. We divide these two topics into fast systems, where specialized FPGAs and ASICs must be used to process the data, and slower systems, where more conventional processing approaches using GPUs, and industry based solutions can be applied.

For low latency systems, we have focused on strategies to shrink networks so that their resource usage and latencies are optimized. In this way, we have shown that compression, quantization, and parallelization are key elements to optimize network design. Furthermore, we have shown that, when considering an algorithm design, processor resources are critical. These ideas are the foundation of hardware-algorithm codesign.

With longer latency systems, we have presented approaches that can use GPUs, FPGAs, and many industry tools. In this scenario, we have presented common strategies including as-a-service computing and directly connected coprocessors. Critical aspects of this design include optimizing scheduling, dynamic batching, and the same hardware-algorithm codesign elements necessary for low latency processing.

Whether at ultra low latencies with specialized processors or longer latencies with existing CPU or GPU configurations, processor technology is a core component of the design and operation of deep learning algorithms. The large speed ups in execution time, despite the complexity of the algorithms, are promising for enhancing computational throughput. The use of regular, repeated matrix multiplication, present in deep learning computation, and reduced bit precision have helped to further improve the computational acceleration. Finally, the continued advancement of processor technology in the coming years opens up the possibility of applying even more sophisticated deep learning algorithms in high energy physics trigger and data acquisition systems.

References

- [1] R. Raina, A. Madhavan and A. Y. Ng, Large-scale deep unsupervised learning using graphics processors, in *Proc. 26th Annual Int. Conf. Machine Learning, ICML '09*, (ACM, New York, NY, USA, 2009), p. 873; doi:10.1145/1553374.1553486.
- [2] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems 25*, eds. F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc., 2012), p. 1097.
- [3] D. C. Cireşan, U. Meier, L. M. Gambardella and J. Schmidhuber, Deep, big, simple neural nets for handwritten digit recognition, *Neural Comput.* **22** (2010) 3207; doi:10.1162/neco_a.00052, arXiv:1003.0358.
- [4] R. H. Dennard *et al.*, Design of ion-implanted MOSFET's with very small physical dimensions, *IEEE J. Solid-State Circuits* **9** (1974) 256; doi:10.1109/JSSC.1974.1050511.
- [5] H. Esmaeilzadeh *et al.*, Dark silicon and the end of multicore scaling, in *Proc. 38th Annual Int. Symp. Computer Architecture, ISCA '11* (ACM, New York, NY, USA, 2011), p. 365; doi:10.1145/2000064.2000108.
- [6] H. Esmaeilzadeh *et al.*, Dark silicon and the end of multicore scaling, in *2011 38th Annual Int. Symp. Computer Architecture (ISCA)* (2011), pp. 365–376.
- [7] Y. Sun, N. B. Agostini, S. Dong and D. Kaeli, Summarizing CPU and GPU design trends with product data (2020); arXiv:1911.11313.
- [8] Y. Shen *et al.*, On-chip optical neuromorphic computing, in *Conf. Lasers and Electro-Optics*, (Optical Society of America, 2016); doi:10.1364/CLEO_SI.2016.SM3E.2.
- [9] CMS Collaboration, Performance of the CMS Level-1 trigger in proton-proton collisions at $\sqrt{s} = 13$ TeV (2020); arXiv:2006.10165.
- [10] ATLAS Collaboration, Operation of the ATLAS trigger system in Run 2 (2020); arXiv:2007.12539.
- [11] CMS Collaboration, The Phase-2 upgrade of the CMS level-1 trigger, CMS Technical Design Report, CERN-LHCC-2020-004, CMS-TDR-021 (2020).
- [12] ATLAS Collaboration, Technical design report for the phase-II upgrade of the ATLAS TDAQ system, ATLAS Technical Design Report, CERN-LHCC-2017-020, ATLAS-TDR-029 (2017).
- [13] T. Head, The LHCb trigger system, *J. Instrum.* **9** (2014) C09015; doi:10.1088/1748-0221/9/09/C09015.
- [14] A. A. Abud, G. Lehmann and R. Sipos, Experience and performance of persistent memory for the dune data acquisition system (2020); arXiv:2011.01341.
- [15] J. Chen *et al.*, Accelerating multigrid-based hierarchical scientific data refactoring on GPUs (2020); arXiv:2007.04457.

- [16] CMS Collaboration, The CMS Experiment at the CERN LHC, *J. Instrum.* **3** (2008) S08004; doi:10.1088/1748-0221/3/08/S08004.
- [17] CMS Collaboration, The CMS trigger system, *J. Instrum.* **12**(01) (2017), P01020; doi:10.1088/1748-0221/12/01/P01020; arXiv:1609.02366.
- [18] ATLAS Collaboration, The ATLAS experiment at the CERN Large Hadron Collider, *J. Instrum.* **3** (2008) S08003; doi:10.1088/1748-0221/3/08/S08003.
- [19] A. Bocci *et al.*, Heterogeneous reconstruction of tracks and primary vertices with the CMS pixel tracker (2020); arXiv:2008.13461.
- [20] J. Duarte, Fast reconstruction and data scouting, in *4th Int. Workshop Connecting the Dots 2018* (2018); arXiv:1808.00902.
- [21] CMS Collaboration, Search for narrow and broad dijet resonances in proton-proton collisions at $\sqrt{s} = 13$ TeV and constraints on dark matter mediators and other new particles, *J. High Energy Phys.* **08** (2018) 130; doi:10.1007/JHEP08(2018)130, arXiv:1806.00843.
- [22] S. Benson, V. Gligorov, M. A. Vesterinen and M. Williams, The LHCb turbo stream, *J. Phys. Conf. Ser.* **664**(08) (2015) 082004; doi:10.1088/1742-6596/664/8/082004.
- [23] ATLAS Collaboration, Search for low-mass dijet resonances using trigger-level jets with the ATLAS detector in pp collisions at $\sqrt{s} = 13$ TeV, arXiv:1804.03496.
- [24] CMS Collaboration, Identification of heavy-flavour jets with the CMS detector in pp collisions at 13 TeV, *J. Instrum.* **13**(05) (2018) P05011; doi: 10.1088/1748-0221/13/05/P05011; arXiv:1712.07158.
- [25] CMS Collaboration, M. Stoye, *et al.*, DeepJet: Generic physics object based jet multiclass classification for LHC experiments, in *Deep Learning for Physical Sciences Workshop at the 31st Conf. Neural Information Processing Systems (NeurIPS)* (2017).
- [26] CMS Collaboration, Identification of heavy, energetic, hadronically decaying particles using machine-learning techniques, *J. Instrum.* **15**(06) (2020) P06005; doi:10.1088/1748-0221/15/06/P06005; arXiv:2004.08262.
- [27] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (2016) p. 770; arXiv:1512.03385; doi:10.1109/CVPR.2016.90.
- [28] A. Vaswani *et al.*, Attention is all you need, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon *et al.* (Curran Associates, Inc., 2017), pp. 5998–6008; arXiv:1706.03762.
- [29] T. B. Brown *et al.*, Language models are few-shot learners (2020); arXiv:2005.14165.
- [30] E. A. Moreno *et al.*, JEDI-net: a jet identification algorithm based on interaction networks, *Eur. Phys. J. C* **80** (2020) 58; doi:10.1140/epjc/s10052-020-7608-4; arXiv:1908.05318.
- [31] E. A. Moreno *et al.*, Interaction networks for the identification of boosted $H \rightarrow b\bar{b}$ decays, *Phys. Rev. D* **102** (2020) 012010; doi:10.1103/PhysRevD.102.012010; arXiv:1909.12285.

- [32] H. Qu and L. Gouskos, ParticleNet: Jet tagging via particle clouds, *Phys. Rev. D* **101**(5) (2020) 056019; doi:10.1103/PhysRevD.101.056019; arXiv:1902.08570.
- [33] X. Ju *et al.*, Graph neural networks for particle reconstruction in high energy physics detectors, in *Machine Learning and the Physical Sciences Workshop at the 33rd Annual Conference on Neural Information Processing Systems* (2019); arXiv:2003.11603.
- [34] A. Caulfield *et al.*, A cloud-scale acceleration architecture, in *49th Annual IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2016, pp. 1–13, doi: 10.1109/MICRO.2016.7783710.
- [35] A. M. Caulfield *et al.*, Configurable clouds, in *Micro Volume 37: Issue: 3* (IEEE, 2017), p. 52.
- [36] J. Fowers *et al.*, Inside Project Brainwave’s cloud-scale, real-time AI processor, *IEEE Micro* **39**(3) (2019) 20; doi:10.1109/MM.2019.2910506.
- [37] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.* **9** (1997) 1735; doi:10.1162/neco.1997.9.8.1735.
- [38] D. O’Loughlin *et al.*, Xilinx vivado high level synthesis: Case studies, in *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CICT 2014). 25th IET Year*, (IEEE, 2014), p. 352; doi:10.1049/cp.2014.0713.
- [39] R. Kastner, J. Matai and S. Neuendorffer, Parallel programming for FPGAs (2018); arXiv:1805.03648.
- [40] Y. Cheng, D. Wang, P. Zhou and T. Zhang, Model compression and acceleration for deep neural networks: The principles, progress, and challenges, *IEEE Signal Processing Magazine* **35**(1) (2011) 126–136 doi:10.1109/MSP.2017.2765695; arXiv:1710.09282.
- [41] L. Deng *et al.*, Model compression and hardware acceleration for neural networks: A comprehensive survey, *Proc. IEEE* **108**(4) (2020) 485; doi:10.1109/JPROC.2020.2976475.
- [42] S. Han, H. Mao and W. J. Dally, Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding, in *4th Int. Conf. Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2–4, 2016*, eds. Y. Bengio and Y. LeCun (2016); arXiv:1510.00149.
- [43] Y. Lecun *et al.*, Optimal brain damage, in *Advances in Neural Information Processing Systems (NIPS 1989)*, Denver, CO, ed. D. Touretzky, Vol. 2 (Morgan Kaufmann, 1990).
- [44] C. Louizos, M. Welling and D. P. Kingma, Learning sparse neural networks through l_0 regularization (2017); arXiv:1712.01312.
- [45] R. Rigamonti, A. Sironi, V. Lepetit and P. Fua, Learning separable filters, in *2013 IEEE Conf. Computer Vision and Pattern Recognition* (2013), pp. 2754–2761; doi:10.1109/CVPR.2013.355.
- [46] E. L. Denton *et al.*, Exploiting linear structure within convolutional networks for efficient evaluation, in *Advances in Neural Information Processing Systems 27*, eds. Z. Ghahramani *et al.* (Curran Associates, Inc., 2014), pp. 1269–1277.

- [47] M. Jaderberg, A. Vedaldi and A. Zisserman, Speeding up convolutional neural networks with low rank expansions, in *Proc. British Machine Vision Conf.* (BMVA Press, 2014); arXiv:1405.3866; doi:10.5244/C.28.88.
- [48] M. Denil *et al.*, Predicting parameters in deep learning, in *Advances in Neural Information Processing Systems 26*, eds. C. J. C. Burges *et al.* (Curran Associates, Inc., 2013), pp. 2148–2156.
- [49] T. N. Sainath *et al.*, Low-rank matrix factorization for deep neural network training with high-dimensional output targets, in *2013 IEEE Int. Conf. Acoustics, Speech and Signal Processing* (2013), pp. 6655–6659; doi: 10.1109/ICASSP.2013.6638949.
- [50] T. S. Cohen and M. Welling, Group equivariant convolutional networks, in *The 33rd Int. Conf. Machine Learning (ICML 2016)* (2016); arXiv:1602.07576.
- [51] C. Buciluă, R. Caruana and A. Niculescu-Mizil, Model compression, in *Proc. 12th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, KDD '06* (ACM, New York, NY, 2006), pp. 535–541; doi:10.1145/1150402.1150464.
- [52] A. Y. Ng, Feature selection, l_1 vs. l_2 regularization, and rotational invariance, in *Proc. 21st Int. Conf. Machine Learning, ICML '04* (ACM, New York, NY, 2004), p. 78; doi:10.1145/1015330.1015435.
- [53] D. Blalock, J. J. G. Ortiz, J. Frankle and J. Gutttag, What is the state of neural network pruning?, in *4th Conf. Machine Learning and Systems* (2020); arXiv:2003.03033.
- [54] J. Frankle and M. Carbin, The lottery ticket hypothesis: Training pruned neural networks, in *7th Int. Conf. Learning Representations* (2019); arXiv:1803.03635.
- [55] A. Renda, J. Frankle and M. Carbin, Comparing rewinding and fine-tuning in neural network pruning, in *8th Int. Conf. Learning Representations* (2020); arXiv:2003.02389.
- [56] C. N. Coelho Jr. *et al.*, Ultra low-latency, low-area inference accelerators using heterogeneous deep quantization with QKERAS and hls4ml (2020); arXiv:2006.10159.
- [57] Google, QKeras preprint (2020); <https://github.com/google/qkeras>.
- [58] M. Blott *et al.*, FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks, *ACM Trans. Reconfigurable Technol. Syst.* **11** (2018); doi:10.1145/3242897; arXiv:1809.04570.
- [59] Alessandro, G. Franco and N. Fraser, Xilinx/brevitas: bnn-pynq-r1 (2020); doi:10.5281/zenodo.4020996.
- [60] CMS Collaboration, Boosted decision trees in the level-1 muon endcap trigger at CMS, *J. Phys. Conf. Ser.* **1085**(4) (2018) 042042; doi:10.1088/1742-6596/1085/4/042042.
- [61] J. Ortiz Arciniega, F. Carrió and A. Valero, FPGA implementation of a deep learning algorithm for real-time signal reconstruction in particle detectors under high pile-up conditions, *J. Instrum.* **14**(09) (2019) P09002; doi: 10.1088/1748-0221/14/09/P09002; arXiv:1903.02439.

- [62] J. Duarte *et al.*, Fast inference of deep neural networks in FPGAs for particle physics, *J. Instrum.* **13** (2018) P07027; doi:10.1088/1748-0221/13/07/P07027; arXiv:1804.06913.
- [63] V. Loncar *et al.*, Compressing deep neural networks on FPGAs to binary and ternary precision with `hls4ml` (2020); arXiv:2003.06308.
- [64] S. R. Qasim, J. Kieseler, Y. Iiyama and M. Pierini, Learning representations of irregular particle-detector geometry with distance-weighted graph networks, *Eur. Phys. J. C* **79** (2019) 608; doi:10.1140/epjc/s10052-019-7113-9; arXiv:1902.07987.
- [65] Y. Iiyama *et al.*, Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics, *Front. Big Data* (2020); doi:10.3389/fdata.2020.598927; arXiv:2008.03601.
- [66] A. Heintz *et al.*, Accelerated charged particle tracking with graph neural networks on FPGAs, in *3rd Machine Learning and the Physical Sciences Workshop at the 34th Annual Conf. Neural Information Processing Systems* (2020); arXiv:2012.01563.
- [67] S. Summers *et al.*, Fast inference of boosted decision trees in FPGAs for particle physics, *J. Instrum.* **15** (2020) P05026; doi:10.1088/1748-0221/15/05/P05026; arXiv:2002.02534.
- [68] A reconfigurable neural network ASIC for detector front-end data compression, in *22nd IEEE Real Time Conf.* (2020).
- [69] M. Wielgosz and M. Karwatowski, Mapping neural networks to FPGA-based IoT devices for ultra-low latency processing, *Sensors* **19** (2019) 2981.
- [70] J. Decaluwe, B. Allard and C. L. Felton, MyHDL (2020); <http://www.myhdl.org/>.
- [71] E. M. Metodiev, B. Nachman and J. Thaler, Classification without labels: Learning from mixed samples in high energy physics, *J. High Energy Phys.* **10** (2017) 174; doi:10.1007/JHEP10(2017)174; arXiv:1708.02949.
- [72] T. Heimel, G. Kasieczka, T. Plehn and J. M. Thompson, QCD or What?, *SciPost Phys.* **6**(3) (2019) 030; doi:10.21468/SciPostPhys.6.3.030; arXiv:1808.08979.
- [73] J. H. Collins, K. Howe and B. Nachman, Anomaly detection for resonant new physics with machine learning, *Phys. Rev. Lett.* **121**(24) (2018) 241803; doi:10.1103/PhysRevLett.121.241803; arXiv:1805.02664.
- [74] M. Farina, Y. Nakai and D. Shih, Searching for new physics with deep autoencoders, *Phys. Rev. D* **101**(7) (2020) 075021; doi:10.1103/PhysRevD.101.075021; arXiv:1808.08992.
- [75] O. Cerri *et al.*, Variational autoencoders for new physics mining at the large hadron collider, *J. High Energy Phys.* **05** (2019) 036; doi:10.1007/JHEP05(2019)036; arXiv:1811.10276.
- [76] J. H. Collins, K. Howe and B. Nachman, Extending the search for new resonances with machine learning, *Phys. Rev. D* **99**(1) (2019) 014038; doi:10.1103/PhysRevD.99.014038; arXiv:1902.02634.
- [77] O. Knapp *et al.*, Adversarially learned anomaly detection on CMS open data: re-discovering the top quark, preprint (2020); arXiv:2005.01598.

- [78] T. Cheng *et al.*, Variational autoencoders for anomalous jet tagging, preprint (2020); arXiv:2007.01850.
- [79] A. Andreassen, B. Nachman and D. Shih, Simulation assisted likelihood-free anomaly detection, *Phys. Rev. D* **101**(9) (2020) 095004; doi:10.1103/PhysRevD.101.095004; arXiv:2001.05001.
- [80] B. Nachman and D. Shih, Anomaly detection with density estimation, *Phys. Rev. D* **101** (2020) 075042; doi:10.1103/PhysRevD.101.075042; arXiv:2001.04990.
- [81] S. E. Park *et al.*, Quasi anomalous knowledge: Searching for new physics with embedded knowledge, preprint (2020); arXiv:2011.03550.
- [82] T. Boltz *et al.*, Accelerating machine learning for machine physics (an AMALEA-project at KIT), in *Proc. Int. Conf. Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19)*, No. 17 (JACoW Publishing, Geneva, Switzerland, 2020), p. 781; doi:10.18429/JACoW-ICALEPCS2019-TUCPL06.
- [83] E. Hubbard *et al.*, Booster Synchrotron, Fermilab Technical Memo, 1 (1973); doi:10.2172/1155286.
- [84] J. S. John *et al.*, Real-time artificial intelligence for accelerator control: A study at the Fermilab Booster (2020); arXiv:2011.07371.
- [85] V. François-Lavet *et al.*, An introduction to deep reinforcement learning, *Found. Trends Mach. Learn.* **11**(3–4) (2018) 219; doi:10.1561/22000000071.
- [86] B. H. Denby, Neural networks and cellular automata in experimental high-energy physics, *Comput. Phys. Commun.* **49** (1988) 429; doi:10.1016/0010-4655(88)90004-5.
- [87] D. Cutts *et al.*, The Use of neural networks in the D0 data acquisition system, *IEEE Trans. Nucl. Sci.* **36** (1989) 1490–1493; doi:10.1109/23.41089.
- [88] B. H. Denby *et al.*, Neural networks for triggering, *IEEE Trans. Nucl. Sci.* **37** (1990) 248; doi:10.1109/23.106627.
- [89] S. Amendolia and B. Denby, Ongoing approaches to the trigger problem using neural networks, *Conf. Proc. C* **901004** (1990) 129.
- [90] L. Lonnblad, C. Peterson and T. Rognvaldsson, Finding gluon jets with a neural trigger, *Phys. Rev. Lett.* **65** (1990) 1321; doi:10.1103/PhysRevLett.65.1321.
- [91] S. Amendolia, Neural networks for trigger, in *1992 CERN School of Computing (15th)* (1992), p. 135.
- [92] B. H. Denby, Status of neural net triggers at Fermilab Tevatron, *Conf. Proc. C* **9201131** (1992) 361.
- [93] D0 Collaboration, Tau identification at Fermilab D0 and higgs searches using taus, *Nucl. Phys. B Proc. Suppl.* **189** (2009) 338–343; doi:10.1016/j.nuclphysbps.2009.03.055.
- [94] V. Gligorov and M. Williams, Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree, *J. Instrum.* **8** (2013) P02013; doi: 10.1088/1748-0221/8/02/P02013; arXiv:1210.6861.
- [95] LHCb Collaboration, Design and performance of the LHCb trigger and full real-time reconstruction in Run 2 of the LHC, *J. Instrum.* **14**(04) (2019) P04013; doi:10.1088/1748-0221/14/04/P04013; arXiv:1812.10790.

- [96] T. Ciodaro, D. Deva, J. de Seixas and D. Damazio, Online particle detection with neural networks based on topological calorimetry information, *J. Phys. Conf. Ser.* **368** (2012) 012030; doi:10.1088/1742-6596/368/1/012030.
- [97] ATLAS Collaboration, Performance of electron and photon triggers in ATLAS during LHC Run 2, *Eur. Phys. J. C* **80**(1) (2020) 47; doi:10.1140/epjc/s10052-019-7500-2; arXiv:1909.00761.
- [98] D. Guest *et al.*, Jet flavor classification in high-energy physics with deep neural networks, *Phys. Rev. D* **94**(11) (2016), 112002; doi:10.1103/PhysRevD.94.112002; arXiv:1607.08633.
- [99] ATLAS Collaboration, Performance of the ATLAS trigger system in 2015, *Eur. Phys. J. C* **77**(5) (2017) 317; doi:10.1140/epjc/s10052-017-4852-3; arXiv:1611.09661.
- [100] CMS Collaboration, Performance of Electron Reconstruction and Selection with the CMS Detector in Proton-Proton Collisions at $\sqrt{s} = 8$ TeV, *J. Instrum.* **10**(06) (2015), P06005; doi:10.1088/1748-0221/10/06/P06005; arXiv:1502.02701.
- [101] ALICE Collaboration, Real-time data processing in the ALICE High Level Trigger at the LHC, *Comput. Phys. Commun.* **242** (2019) 25–48, doi:10.1016/j.cpc.2019.04.011; arXiv:1812.08036.
- [102] R. Aaij *et al.*, Allen: A high level trigger on GPUs for LHCb, *Comput. Softw. Big Sci.* **4**(1) (2020) 7; doi:10.1007/s41781-020-00039-7; arXiv:1912.09161.
- [103] D. Vom Bruch, Real-time data processing with GPUs in high energy physics, *J. Instrum.* **15**(06) (2020) C06010; doi:10.1088/1748-0221/15/06/C06010; arXiv:2003.11491.
- [104] D. Funke *et al.*, Parallel track reconstruction in CMS using the cellular automaton approach, *J. Phys. Conf. Ser.* **513** (2014) 052010; doi:10.1088/1742-6596/513/5/052010.
- [105] F. Pantaleo *et al.*, Development of a phase-II track trigger based on GPUs for the CMS experiment, in *2015 IEEE Nuclear Science Symposium and Medical Imaging Conference* (2016), p. 7581775; doi:10.1109/NSSMIC.2015.7581775.
- [106] ATLAS Collaboration, Multi-threaded algorithms for GPGPU in the ATLAS High Level Trigger, *J. Phys. Conf. Ser.* **898** (2017) 032003 doi:10.1088/1742-6596/898/3/032003.
- [107] G. Lamanna, Almagest, a new trackless ring finding algorithm, *Nucl. Instrum. Methods Phys. Res. A* **766** (2014) 241; doi:10.1016/j.nima.2014.05.073.
- [108] C. Färber, R. Schwemmer, J. Machen and N. Neufeld, Particle identification on an FPGA accelerated compute platform for the LHCb upgrade, in *20th IEEE-NPSS Real Time Conference*, (2016) p. 1; doi:10.1109/RTC.2016.7543150.
- [109] Mu3e Collaboration, Online data reduction using track and vertex reconstruction on GPUs for the Mu3e experiment, *Eur. Phys. J. Web Conf.* **150** (2017) 00013; doi:10.1051/epjconf/201715000013.

- [110] R. Ammendola *et al.*, Real-time heterogeneous stream processing with NaNet in the NA62 experiment, *J. Phys. Conf. Ser.* **1085**(3) (2018) 032022; doi:10.1088/1742-6596/1085/3/032022.
- [111] A. Reuther *et al.*, Survey and benchmarking of machine learning accelerators, in *2019 IEEE High Performance Extreme Computing Conference (HPEC)* (IEEE, 2019); arXiv:1908.11348; doi:10.1109/hpec.2019.8916327.
- [112] A. Reuther *et al.*, Survey of machine learning accelerators, in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–12; doi:10.1109/HPEC43674.2020.9286149.
- [113] A. Li *et al.*, Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect, *IEEE Trans. Parallel Distributed Systems* **31**(1) (2020) 94–110; doi:10.1109/TPDS.2019.2928289; arXiv:1903.04611.
- [114] R. Neugebauer *et al.*, Understanding pcie performance for end host networking, *SIGCOMM '18* (ACM, New York, NY, 2018), p. 327; doi:10.1145/3230543.3230560.
- [115] N. Eskandari, N. Tarafdar, D. Ly-Ma and P. Chow, A modular heterogeneous stack for deploying fpgas and cpus in the data center, in *FPGA '19*, (ACM, New York, NY, 2019), p. 262; doi:10.1145/3289602.3293909.
- [116] J. Duarte *et al.*, FPGA-accelerated machine learning inference as a service for particle physics computing, *Comput. Softw. Big Sci.* **3**(1) (2019) 13; doi: 10.1007/s41781-019-0027-2; arXiv:1904.08986.
- [117] Nvidia, Triton inference server [software], <https://docs.nvidia.com/deeplearning/sdk/triton-inference-server-guide/docs/index.html>, 2019. v1.8.0 (accessed 2020-02-17).
- [118] J. Krupa *et al.*, GPU coprocessors as a service for deep learning inference in high energy physics, preprint (2020); arXiv:2007.10359.
- [119] D. S. Rankin *et al.*, FPGAs-as-a-service toolkit (FaaSST), in *2020 IEEE/ACM Int. Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)* (2020); arXiv:2010.08556.
- [120] M. Wang *et al.*, GPU-accelerated machine learning inference as a service for computing in neutrino experiments, preprint (2020); arXiv:2009.04509.
- [121] V. Kuznetsov, L. Giommi and D. Bonacorsi, MLaaS4HEP: Machine learning as a service for HEP (2020); arXiv:2007.14781.
- [122] Google, gRPC [software], <https://grpc.io/>, 2018. v1.19.0 (accessed 2020-02-17).
- [123] Intel, Thread building blocks [software], <https://www.threadingbuildingblocks.org>, 2018. 2018_U1 (accessed 2019-02-11).
- [124] A. Bocci *et al.*, Bringing heterogeneity to the CMS software framework, in *24th Int. Conf. Computing in High Energy and Nuclear Physics.* (2020); arXiv:2004.04334.
- [125] G. Neubig, Y. Goldberg and C. Dyer, On-the-fly operation batching in dynamic computation graphs, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon *et al.* (Curran Associates, Inc., 2017), p. 3971; arXiv:1705.07860.

- [126] A. Massironi, V. Khristenko and M. D'Alfonso, Heterogeneous computing for the local reconstruction algorithms of the CMS calorimeters, *J. Phys. Conf. Ser.* **1525** (2020) 012040.
- [127] B. Holzman *et al.*, HEPCloud, a new paradigm for HEP facilities: CMS Amazon Web Services investigation, *Comp. Soft. Big Sci.* **1** (2017); doi: 10.1007/s41781-017-0001-9; arXiv:1710.00100.
- [128] M. Altunay *et al.*, Intelligently-automated facilities expansion with the HEPCloud decision engine, preprint (2018); arXiv:1806.03224.
- [129] P. Mhashikar *et al.*, HEPCloud, an elastic hybrid HEP facility using an intelligent decision support system, *Eur. Phys. J. Web Conf.* **214** (2019) 03060; doi:10.1051/epjconf/201921403060; arXiv:1904.08988.
- [130] L. Michel, Interaction between four half spin particles and the decay of the μ meson, *Proc. Phys. Soc. A* **63** (1950) 514; doi:10.1088/0370-1298/63/5/311.

This page intentionally left blank

Part V

Detector Data Reconstruction

This page intentionally left blank

Chapter 10

End-to-End Analyses Using Image Classification

Adam Aurisano* and Leigh H. Whitehead†

**Department of Physics, University of Cincinnati,
Cincinnati, OH 45221, USA
aurisaam@ucmail.uc.edu*

*†Cavendish Laboratory, University of Cambridge,
Cambridge, CB3 0HE, UK
leigh.howard.whitehead@cern.ch*

End-to-end analyses of data from high-energy physics experiments using machine and deep learning techniques have emerged in recent years. These analyses use deep learning algorithms to go directly from low-level detector information directly to high-level quantities that classify the interactions. The most popular class of algorithms for these analyses are convolutional neural networks that operate on experimental data formatted as images. End-to-end analyses skip stages of the traditional workflow that includes the reconstruction of particles produced in the interactions, and as such are not limited by efficiency losses and sources of inaccuracy throughout the event reconstruction process. In many cases, deep learning end-to-end analyses have been shown to have significantly increased performance compared to previous state-of-the-art methods.

1. Introduction

End-to-end analyses take their name from the fact that they use a single algorithm that takes raw or low-level detector data as input and outputs high-level physics information for each event. By definition, these analyses skip most, or all, of the traditional workflow for particle physics analyses. Deep learning approaches are a natural choice for these algorithms that can extract features from the input

data to perform powerful classifications. Inspired by developments in computer vision and image recognition, a popular choice of algorithm is the two-dimensional (2D) convolutional neural network (CNN) [1]. CNNs operate on image-like, lattice-structured inputs produced from raw or low-level detector data to generate predictions of physics-level outputs to classify and describe interactions, such as identified particles and overall event-type classifications.

Section 2 outlines a typical example of the traditional reconstruction and analysis workflow, and Sec. 3 discusses the two primary end-to-end analysis algorithms, CNNs and graph neural networks (GNNs). Use cases for CNNs are presented for lattice-structured experiments in Sec. 4, for non-lattice-structured experiments in Sec. 5, and for time-series data in Sec. 6. Section 7 describes a use case for end-to-end analysis using GNNs. Section 8 details methods for probing the behavior of deep neural networks, and Sec. 9 provides some concluding remarks.

2. Traditional Workflow

The specifics of each analysis workflow can vary widely between different experiments and sub-disciplines within high-energy physics, but they can typically be broken down into four main steps: low-level reconstruction, particle clustering, particle identification, and event classification. It should be noted that some, or all, of these stages could include machine (and deep) learning aspects, such as boosted decision trees (BDT), neural networks and CNNs in order to make important decisions at key points in the workflow. These stages are discussed briefly below.

Low-level reconstruction: In this context, low-level reconstruction refers to the finding of signals from the active detector elements, for example the electronic readout channels from a silicon vertex locator in a collider detector, or the readout wires in a liquid argon time projection chamber (LArTPC). These raw signals are processed and converted in some way to produce *hit* objects that form the basis of further event reconstruction. Each hit represents an energy deposit at a given location in space at a specific time.

Particle clustering: The reconstructed hit objects form the basis of the main particle reconstruction. A set of clustering algorithms are applied to group hits together based on spatial and temporal distance. These clusters are then associated together to build up objects representing each of the individual particles that interacted inside the detector. These objects generally fall into two categories with track- or shower-like topologies. Particles such as muons that lose energy primarily by ionization leave track-like energy deposits in the detectors, whereas particles such as electrons and photons tend to initiate electromagnetic (EM) cascades of particles forming shower-like topologies. The aim is to have a list of fully reconstructed particles at the end of this reconstruction step.

Particle identification: Once the individual particles have been reconstructed, they can be classified as a specific type of particle. The identification of track-like particles typically includes the use of the energy loss per unit length, $\frac{dE}{dx}$, and the track curvature in a magnetic field for determination of its momentum and the sign of the electromagnetic charge. Topological information in particle cascades, often referred to as *jets* or *showers*, is used to identify particles that produce non-track-like energy deposits.

Event classification: At this stage the reconstructed interaction contains all of the reconstructed particles with an attached measured particle type. Full events are built from the individual particles and associations are made between the different particles to give the flow of the interaction. Finally, an overall classification of the full physics interaction is given along with important variables that describe the interaction as a whole, for example the energies of the colliding particles.

3. Deep Learning Approaches

Image-like data has been common in particle physics since its earliest days. In particular, the bubble chamber, invented around 1952 by Donald Glaser, was a key detector technology for decades [2]. These detectors dominated the field because they were reliable, fully active,

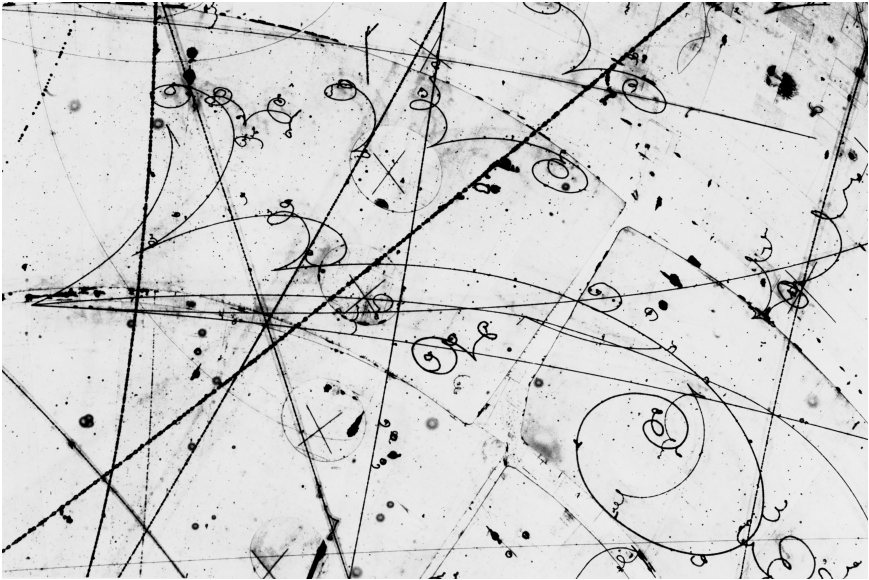


Fig. 1. An example of tracks captured by a bubble chamber. These images were typically manually reconstructed by trained human scanners. Illustration courtesy of Fermilab.

and had high spatial resolution. All of these properties were key in developing our understanding of hadron properties and electroweak unification.

Figure 1 shows a typical image of tracks recorded by a bubble chamber. These chambers would be exposed to charged particle beams, and the resulting tracks would be recorded on photographic film. Trained human scanners reconstructed the decays of particles in the beam manually by visually recognizing and isolating meaningful features like vertices or tracks which the scanners would then physically measure.

The focus on studying rare interactions required larger detectors with a higher data rate making hand-scanning increasingly impractical. This eventually led to the development of technologies that relied on electronic readout which was automatically reconstructed using techniques described in Sec. 2. While the traditional workflow has proven very successful, current and next-generation high-energy

physics experiments can provide very fine details of interactions in comparison to previous experiments. To give an example from neutrino physics, charged-current (CC) ν_μ interactions consist of a muon with accompanying hadronic activity (nucleons and any number of charged and neutral pions). In the MINOS detectors [3], which are relatively coarse-grained due to the use of thick steel plates between scintillator planes, CC ν_μ events looked like a long muon track along with a collection of energy deposits from the hadronic activity at the interaction vertex. NOvA [4], which contains little dead material and is in many ways the successor to the MINOS experiment, provides more detail of the hadronic system and can resolve some of the particles. A next-generation experiment such as DUNE [5], which uses liquid argon time projection technology (LArTPC) [6], begins to approach resolutions similar to bubble chambers. Thus the DUNE detectors will be able to image all of the particles in the hadronic system in fine detail. Figure 2 shows example CC ν_e interactions in each detector and demonstrates graphically the ever-increasing requirements of event reconstruction algorithms to accurately reconstruct the interactions with improving experimental detector resolution. All of the stages of event reconstruction in the traditional workflow are imperfect in terms of both reconstruction efficiency and accuracy. Any mistakes made by the reconstruction algorithms tend to compound through the reconstruction chain and will result in inefficiencies and backgrounds in physics analyses. In addition, each stage of the traditional reconstruction workflow is designed to summarize information about reconstructed features, leading to information

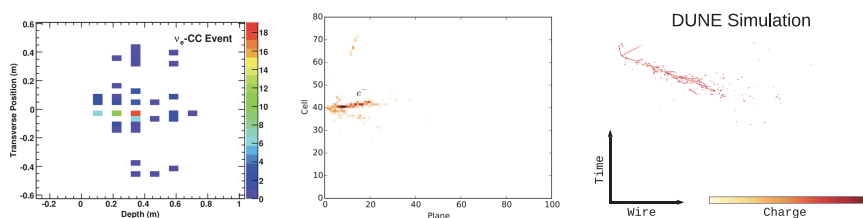


Fig. 2. Example CC ν_e interactions in progressively higher resolution detectors. Left: event display from MINOS from [7]. Center: event display from NOvA from [8]. Right: event display from DUNE, adapted from [9].

loss as summary information from low-level reconstructed objects is combined to form high-level reconstruction objects. Therefore, it is natural to turn to the field of computer vision to find automated approaches to approximate the tasks human scanners performed in previous decades to efficiently use all information collected by the detector.

The inputs to end-to-end deep learning analyses are typically only dependent on the low-level reconstruction and can hence provide powerful analysis-level information without potential errors from the full event reconstruction algorithm chain. However, careful consideration is required for the selection of the training samples for these algorithms to ensure that are not biased due to overfitting or fine-tuning on the training sample. This is particularly important when training on Monte Carlo (MC) simulations based on physics models with associated uncertainties. Two broad categories of deep learning techniques are discussed and demonstrated with examples from various high-energy physics experiments: CNNs and GNNs in Secs. 3.1 and 3.2, respectively.

3.1. *Convolutional neural networks*

The artificial neural network [10], also known as a multilayer perceptron (MLP), is a machine learning algorithm characterized by layers of nodes with defined connections between them. Each node represents a nonlinear function of the sum of all input connections, and each connection is associated with a weight parameter which scales the output of one node to become the input of another. With a suitable selection of weights, usually chosen through a training procedure, an MLP can approximate a wide variety of functions.

Traditional, or fully connected, MLPs consist of nodes arranged in layers where each node in layer $n-1$ is connected to each node in layer n . They have been widely used in high-energy physics as selection functions to learn whether or not a given event is signal or background based on a series of input reconstructed quantities. However, this technique, which sits at the end of the traditional reconstruction workflow, is subject to all potential compounding errors in the event reconstruction, described in Sec. 2.

The CNN [11] is similar to a fully connected MLP except the pattern of connections between nodes on different layers is tightly constrained. This structure was inspired by studies of the visual cortex of cats and monkeys which determined that cells within the visual cortex were activated by specific illumination patterns on regions of the retina known as receptive fields [12–14]. The characteristics of the receptive field were observed to vary for different cells within the cortex, leading to the cells being classified as simple, complex, or hypercomplex. Simple cells are those which are most strongly activated by a static illumination pattern of a specific location, orientation, and shape either on one eye or on corresponding locations on both eyes. Complex cells respond to orientation and shape, but instead of responding to a specific location, they respond to movement of the illumination pattern through a receptive field. Hypercomplex cells additionally respond to the length of an illumination pattern. Hubel and Wiesel hypothesized that complex cells received signals from simple cells, and hypercomplex cells received signals from complex cells. This implies that the mammalian visual cortex analyzes images by using a hierarchical network of cells which have local connections from one layer to the next. In this way, the brain extracts edge features at the lowest layers, it determines directionality in the middle layers, and it finds extents in the highest layers.

CNNs were first used in the 1980s to identify handwritten digits [1], but they did not become widespread until 2012 with the success of AlexNet [15] in identifying images in the ImageNet challenge [16]. The dramatic success of AlexNet has since produced a proliferation of CNN architectures that have improved image classification to super-human levels; however, despite the diversity in architectures, all CNNs share some common structures, which we will detail below.

The structure of CNNs begins from the insight that images can be interpreted as an array of numbers of size $h \times w \times c$, where h and w are the height and width of the image, respectively, in pixels, and c is the number of channels (also known as the depth). For grayscale images, $c = 1$, while for color images, $c = 3$. The value of each array element represents the intensity of the light at the corresponding location and channel.

This array is then passed through a series of layers which perform local operations under the assumption that pixels close to each other are likely to be semantically related. Each layer produces a series of $h \times w \times c$ outputs known as feature maps, with potentially changed sizes of h , w , and c . As feature maps are passed through layers, features represent increasingly more global information as local information from larger regions is combined. The full column of layers learns to automatically extract high-level features which replace the use of hand-crafted features that are typically used in traditional MLPs. The features produced by the final layer are fed into a single layer of a traditional MLP to produce the outputs which approximate the function the network was trained to learn.

Many types of layers have been developed for use in CNNs, but the most important types are convolutional layers and pooling layers. All CNNs contain convolutional layers which directly mimic the concept behind the simple cell in the visual cortex. A simple cell receives inputs from a local region of the retina and weights each input according to a particular excitatory or inhibitory pattern. Convolutional operators mimic this by taking the dot product of the values in a local region of a feature map with a matrix of learnable weights known as a convolutional kernel. This operation is repeated across the entire feature map to produce an output feature map. This operation is closely related to the discrete convolution.

The choice to use the same convolutional kernel to extract features across the full input produces two properties that are responsible for much of the power of CNNs. First, learning a single kernel per output feature map dramatically reduces the number of free parameters learned by the network compared to fully connected layers in a traditional MLP. This makes CNNs easier to train and less susceptible to overtraining. Second, applying the same operation at every local region makes the layer equivariant to translation. That is, if an object is translated in an image, the same features will be produced, just translated within the output feature map. If this property is maintained throughout the network, the efficiency for detecting if a particular object is present in an image will not depend on the exact location of the object in the image. However, scale, rotation,

and the relative positions of objects within the image can still be important.

Pooling layers are optional in CNNs, but they are extremely common. They apply a pooling operator which combines features from a local patch in an irreversible way. The most commonly used pooling operator is max pooling, which outputs the maximum value of the features in a local patch. For example, a 2×2 max pooling layer downsamples 2×2 input pixels to a single pixel with the value of the highest-valued input pixel. This has the effect of removing low significance information and imposing an invariance to small translations. The invariance property reduces the sensitivity of the network to exact positions, including relative positions.

3.2. Graph neural networks

There are many situations where representing experimental data as an image is not a natural or convenient method. For example, experiments with complex geometries may require numerous projections of the data to produce 2D images. Even in experiments where images provide a good data representation, there are cases where only a small fraction of the detector elements are activated for a given event, resulting in images with many empty pixels (see many of the images discussed in Sec. 3.1). This does not necessarily present a problem, but a lot of time can be spent performing convolutions on pixels with zero values, which always gives a zero result regardless of the filter applied.

In the case of complex geometries, a more natural representation may involve considering each detector element as a point in 3D space. Furthermore, considering only those detector elements with a measured signal on an event-by-event basis will provide a more efficient approach for sparse data. A data structure that copes well with these requirements is a graph

$$G = (V, E), \tag{1}$$

where V are a set of vertices, also known, and henceforth referred to, as *nodes*, and E are a set of *edges*. Edges are defined as connections

between nodes such that edge e_{ji} links node v_j to node v_i . This is an example of a *directional* edge since it points from node v_j to node v_i . Edges can also be *undirected*, in which case $e_{ji} = e_{ij}$ and the link is reciprocated. Each node has a number of *features* associated to it that describe its properties.

To form graphs from high-energy physics data, each detector element with a measured energy deposit is added as a graph node. Each node has a number of features, which could include information such as the position of the detector element and the amount of deposited energy. The ability to associate multiple features with each node provides an easy way to incorporate more information into a GNN beyond just position and charge. The edges that link the nodes can be defined in a number of ways, for example using the adjacency of nodes to their neighbors. Each interaction is therefore represented as a connected graph containing all of the recorded energy deposits in the detector.

Graph neural networks [17–21] (GNNs) are neural networks that operate on graphs. Depending on the specific classification task, the GNN can classify nodes, edges or the entire graph. There is a large variety of GNN architectures [22], but they typically use graph-based convolutions to aggregate the features of a node and its neighbors.

3.3. *Network optimization*

Convolutional and graph neural networks, like all machine learning algorithms, need to undergo a training procedure, and the performance of the final algorithm is highly dependent on the quality of the training. The choice of training samples, typically from simulation, is a key consideration to ensure that the algorithm generalizes well, meaning that it performs similarly on data not included in the training sample. There are many other important factors, including the choice of optimizer used to find the minimum of the loss function, and the network *hyperparameters*.

Training samples: The choice of training sample is very important for CNNs and GNNs. These networks typically have of the order

of millions of parameters and therefore need large training samples to be successfully trained and optimized. The vast majority of networks used in end-to-end analysis are trained using simulated data events and the associated truth information is used to provide the target labels. It is very important to ensure that the training sample covers the entire range of possible interactions that could be seen in the samples that the network will be used to classify.

Optimizers: In machine learning, the training process minimizes a loss function that describes how close the prediction is to the true value(s). The loss function exists in a very high-dimensional space and varies as a function of the trainable parameters that form the network model. Gradient descent is the general method for finding the (local) minimum of the loss function, whereby gradients are calculated with respect to each parameter and then parameter values are updated using the negative of the gradients multiplied by a factor called the *learning rate*. Many optimizers are variants of stochastic gradient descent [23] (SGD), a method where the parameter values are updated after each training example (or more commonly, mini-batch of examples). There are a number of different optimizers that are extensions to the standard SGD algorithm that aim to improve performance and ensure robustness, such as ADADELTA [24], RMSProp [25], Adam [26], etc. For a more detailed discussion of optimizers, see, for example, [27].

Hyperparameters: Parameters that cannot be optimized during training are known as hyperparameters. These are usually either parameters controlling the structure of the network itself, like the number and type of layers, or controlling the behavior of the optimizer. One of the most important hyperparameters of the latter type is the learning rate. If it is too large, then the optimizer can fail to find a minimum in the loss function, but if it is too small, then the optimizer can get stuck in a local (and possibly shallow) minimum. More complex approaches involve decaying the learning rate as a function of the training time in order to avoid local minima and fall into the bottom of a deep (or hopefully global) minimum.

4. Convolutional Neural Networks in Lattice-Structured Experiments

Due to the low interaction rate of neutrinos, neutrino detectors are typically large, and since neutrinos are equally likely to interact anywhere within the volume of the detector, neutrino detectors are typically homogeneous. Therefore, many neutrino detectors produce data which can be easily reinterpreted as images. As such, the first uses of CNNs to perform end-to-end analyses in particle physics occurred at neutrino experiments.

In addition, similar detector technologies are used in neutrinoless double beta decay and nuclear physics experiments. The lattice-structured geometries in these experiments lead to commonalities in the approaches used.

Event classification in the NOvA experiment: The NOvA experiment [4] is a long-baseline neutrino experiment designed to measure ν_μ disappearance and ν_e appearance in a beam originally composed of mostly ν_μ . NOvA measures the flavor content and energy spectrum of the neutrino beam at a near and far location using two functionally identical detectors located on the surface and composed of layers of alternating vertical and horizontal liquid-scintillator-filled PVC cells. The alternating structure provides two orthogonal views of the 3D pattern of energy deposits produced by charged particles traversing the detector projected on the x - z and y - z planes.

To perform oscillation analyses, it is critical to be able to separate events into charged- and neutral-current (NC) interactions, and in the CC case, events must further be separated according to flavor. Since NOvA is on the surface, the cosmic ray flux is large, so it is also necessary to distinguish between cosmic ray and neutrino events. A CNN algorithm was developed to achieve this separation [8]. The input to the CNN consists one image of $100 \text{ planes} \times 80 \text{ cells}$ for both views. These smaller images are extracted from the much larger images representing the full detector by performing a clustering of

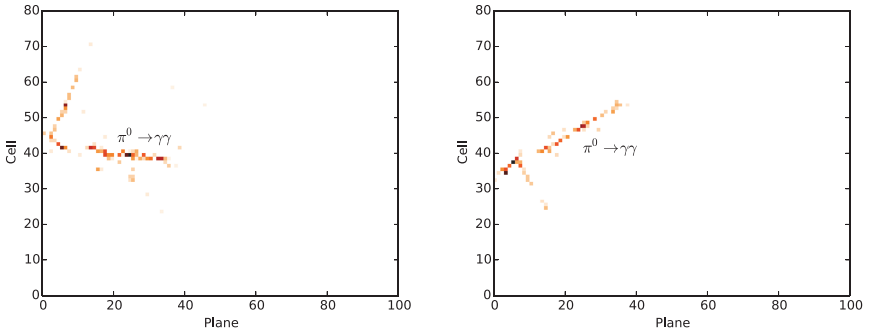


Fig. 3. Example of a pair of input pixel maps for a neutral current interaction containing an electromagnetic shower produced by a π^0 . Each view corresponds to either a projection on the x - z plane (left) or the y - z plane (right). Figures reproduced from [8].

energy deposits in space and time. This clustering is the only reconstruction performed on NOvA data prior to feeding it into the CNN. An example of these inputs for an NC event is shown in Fig. 3.

The NOvA network is based on a modified GoogLeNet architecture [28]. The hallmark of this network is a network-in-network design [29] where miniature CNNs consisting of several convolutional layers operating in parallel with a variety of kernel sizes form a repeatable “Inception module”. Feature maps from each parallel branch in the module are merged together and resampled using a 1×1 convolutional layer.

Since the input NOvA pixel maps consist of two views of the same data sharing one common axis and one different axis, there is no guarantee that the same pixel location on the two views are physically correlated. Therefore, each view is processed separately by two branches of the CNN containing three inception modules. After this stage, the resulting feature maps are sufficiently abstract that they can be concatenated and passed through one final inception module. Separate outputs of the network predict if an event is a cosmic ray, or if it is a neutrino, its flavor and interaction type. Figure 4 shows the performance of the CC ν_e and CC ν_μ classification outputs. The network produces a 40% increase in CC ν_e selection

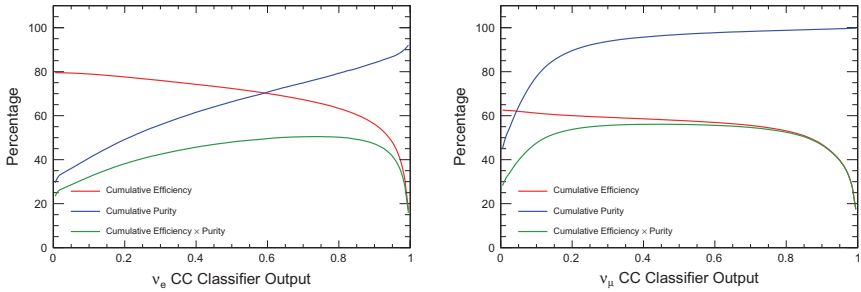


Fig. 4. Classifier efficiency (red), purity (blue), and their product (green) for CC ν_e (left) and CC ν_μ (right) interactions. Figures reproduced from [8].

efficiency over previously used traditional selection techniques with no loss in purity.

This network is the first use of a CNN in a published particle physics analysis [30, 31]. Due to its versatility, the network presented here, as well as subsequent improved networks, has formed the basis of all NOvA oscillation analyses.

Event localization and classification in the MicroBooNE experiment: The MicroBooNE detector [32] is a LArTPC located on the surface on the Fermilab campus with three wire readout planes that collect ionization charge liberated by charged particles traversing the detector medium, and a photon detection system to measure scintillation light. CNN-based algorithms were used, for the first time in a LArTPC experiment, to perform classification of cosmic ray and neutrino interactions [33]. A number of studies were performed, two of which are discussed below.

The first CNN algorithm using the charge information from a single readout plane was developed to perform the event classification and find the bounding box containing the neutrino interaction. The network uses a hybrid architecture based on AlexNet [15] and Faster R-CNN [34]. An example of a correctly classified CC ν_μ interaction is shown in Fig. 5 with a large overlap of the true (yellow) and predicted (red) bounding boxes, and the distribution of the neutrino classification score is shown on the right.

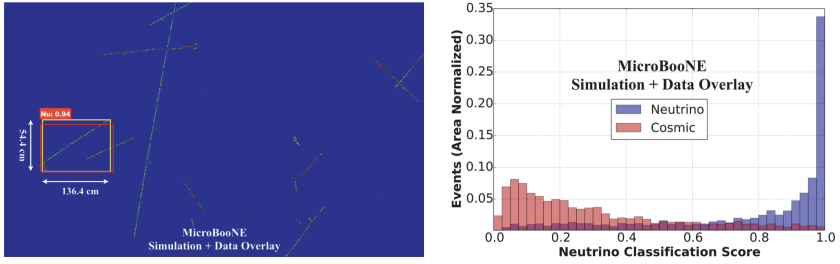


Fig. 5. Left: identified neutrino interaction with a bounding box compared between prediction (red) and truth (yellow). The true bounding box is defined as the smallest region containing all true charge depositions from the simulation, and the prediction is the output of the CNN. Right: the neutrino classification score for cosmic and neutrino interactions. Figures reproduced from [33].

The second CNN algorithm used all three readout views plus the photon detector system. The input for the network consists of a 768×768 pixel depth-12 image, formed from three depth four images (one for each of the three readout views). The components of the depth four images are the following features for a given wire and time: the deposited charge; a binary map of charge deposits consistent with minimum ionizing particles, such as muons and pions; a binary map of charge deposits consistent with heavily ionising particles, such as protons; and a deposited charge map weighted by the distance of the charge detection point from the averaged light collection point from the photon detectors. The last of these images is used to help the CNN find the most important region of the detector where the neutrino interaction is most likely to have occurred. The network architecture was based on the ResNet [35], using three convolutional layers followed by nine ResNet modules, with two final outputs that give scores for the interaction to be a cosmic ray or neutrino event. Figure 6 shows the performance of the classifier for simulated neutrino interactions overlaid with cosmic rays from data. A comparison with the distribution on the right-hand side of Fig. 5 shows that including of all the detector information gives a significant improvement in the classification, as expected.

Event and particle content classification in the DUNE experiment: The Deep Underground Neutrino Experiment (DUNE) [5]

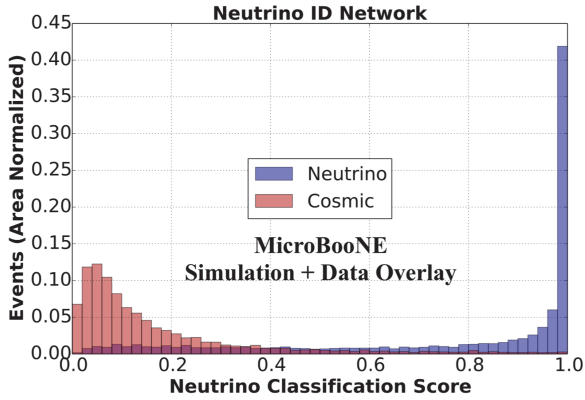


Fig. 6. The neutrino classifier distribution for cosmic and neutrino interactions using all detector information. Figure reproduced from [33].

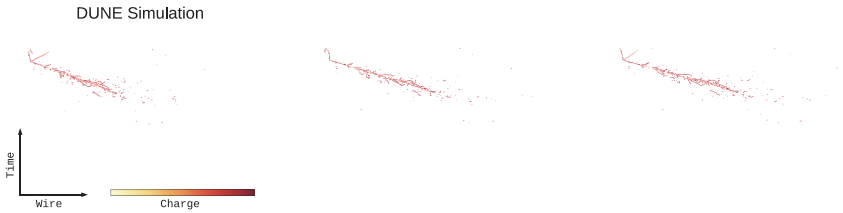


Fig. 7. An example of a simulated CC ν_e interaction in the DUNE far detector, shown in each of the three independent readout views. Figure adapted from [9].

is a next-generation long-baseline neutrino oscillation experiment. The detectors will use LArTPC technology with three wire readout planes. The data from each of these three readout planes can be visualized as a 2D image with coordinates of wire number and time, as shown for a CC ν_e interaction in Fig. 7, where the time coordinate is common between the three images. The 500×500 pixel images are cropped around the neutrino interactions. The pixel values represent the reconstructed charge measured on a given wire at a given time. The DUNE CNN algorithm [9], also known as the CVN, has an architecture based on the SE-ResNet-34 [35–37]. The initial layers of the network are divided into three branches, one for each of the input images, and seven convolutional layers are applied before the

branches are merged together. The final fully connected layer has a number of different outputs but the primary one is designed to identify the type of neutrino interaction. A number of the other outputs from the CVN provide numbers of different final-state particles visible in the neutrino interactions, including protons, charged pions and neutral pions.

The neutrino flavor output of the CVN contains four nodes and returns a score for the event to originate from one of four broad categories: CC ν_μ , CC ν_e , CC ν_τ or NC. These output scores provide very powerful neutrino event classification, as shown by the CC ν_e score and CC ν_μ score distributions on the left and right of Fig. 8, respectively. These scores are used to produce event selections for the neutrino oscillation sensitivities described in detail in [38]. CC ν_e ($\bar{\nu}_e$) interactions are selected with over 90% (95%) efficiency.

Going beyond neutrino flavor classification, the particle counting outputs of the DUNE CVN aim to select interactions with specific final state particles. The output scores for each output are in the range zero to one meaning that a compound score for a given topology can be formed by multiplying the component scores. For example, a score for an event to be a CC ν_μ interaction with a single proton in

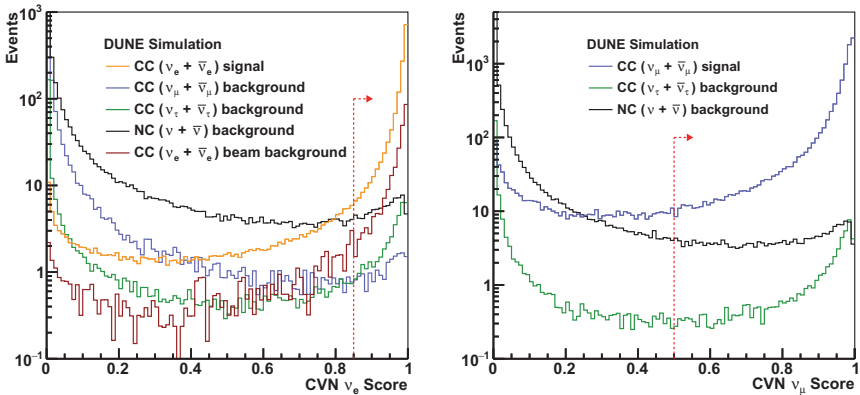


Fig. 8. The output score distributions from the DUNE CVN for the CC ν_e (left) and CC ν_μ (right) hypotheses, shown for the various neutrino flux components. The red arrows correspond to the cut values used in the DUNE event selections [38]. Figure reproduced from [9].

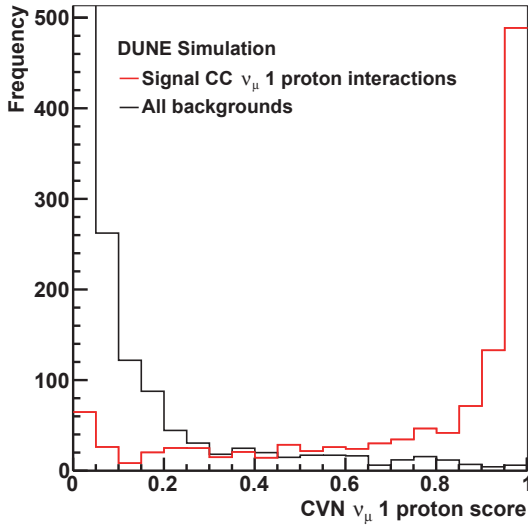


Fig. 9. The score for CC ν_μ interactions with a single final-state proton from the DUNE CVN. Figure adapted from [9].

the hadronic final-state system can be written as follows:

$$S(\text{CC } \nu_\mu 1 \text{ proton}) = S(\text{CC } \nu_\mu) S(1 \text{ proton}) S(0\pi^\pm) S(0\pi^0).$$

Figure 9 shows the distribution of $S(\text{CC } \nu_\mu 1 \text{ proton})$ for signal and all background interactions, providing a proof-of-principle for the selection of specific interaction topologies in the DUNE detectors. The ability to sub-divide the neutrino event selections can improve the analysis sensitivity as some events have better energy resolution and lower systematic uncertainties than others.

Event energy and position reconstruction in EXO-200:

EXO-200 is an experiment searching for neutrino-less double beta decay [39]. The detector is a liquid xenon TPC, similar to LArTPC technology, consisting of two drift regions, and each drift volume has two wire readout planes (one induction and one collection) each consisting of 38 wires. Each drift volume also has an array of 37 large-area avalanche photodiodes (APDs) that collect scintillation light. Two different CNN-based algorithms [40] have been developed to find the energy and position of candidate events, respectively.

The first of these CNNs uses information from the charge readout wires, and the second uses the signals from the APDs.

In the charge-based algorithm, an image is constructed from the charge detected on each of the 76 collection wires in 1024 time samples, resulting in a (1024×76) pixel image. The CNN architecture contains six convolutional layers, each followed by a max pooling layer. A series of three fully connected layers culminates in a single output node that predicts the energy of the interaction. A small improvement, typically a few percent, is seen in the energy resolution at a number of different energies compared to the traditional reconstruction methods.

The second algorithm aims to reconstruct the position of an event using the distribution of scintillation light detected in the APDs. The (350×74) pixel images are produced from waveforms with 350 time samples for each of the 74 APDs. The CNN architecture chosen consisted of four convolutional layers interspersed with max pooling layers that feed into a three layer MLP, with the final layer returning the (x, y, z) position of the interaction. A novel approach is used to generate the training sample using data labeled with the reconstructed position from the wire readout system, since the charge and light readout systems are independent. This approach will therefore minimize the dependence of CNN performance on different physics models. The data were recorded in a number of calibration runs with different radioactive isotopes. Figure 10 shows that the algorithm works well when applied to the calibration data samples.

Improving generalization with data from the AT-TPC: The active-target time projection chamber (AT-TPC) [41] is a detector at the National Superconducting Cyclotron Laboratory at Michigan State University. It is similar to other TPCs discussed here, such as MicroBooNE, DUNE, and EXO-200, but with a few notable differences. Instead of being filled with a liquid noble gas, the AT-TPC is filled with a gas which serves as both the target for the experiment and the drift medium. The apparatus is placed in a magnetic field so that tracks travel in curved trajectories based on their momentum, and the readout system consists of pads, rather than wires, so that

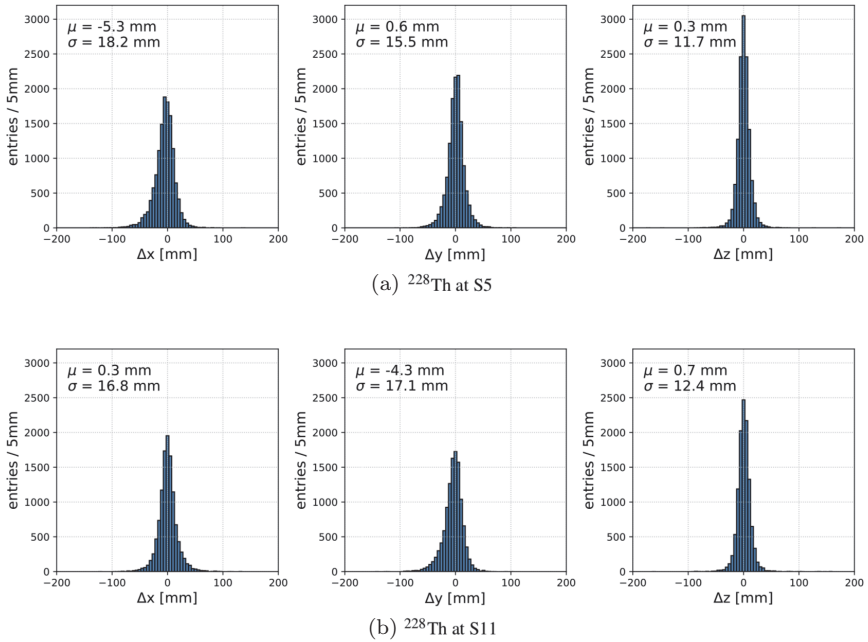


Fig. 10. The position residual and resolution for two calibration data samples. The small bias seen in the top left and bottom middle distributions can be attributed to geometric effects. The calibration source for the data shown in (a) is located near to the edge of the detector in x , and the source used for (b) is near the edge of the detector in y . Figure reproduced from [40].

the readout is inherently three dimensional. The AT-TPC is designed to hold a variety of gases to allow for the study of low-energy nuclear reactions with low rates.

Since many experiments run at the AT-TPC, each for a short time and generating large volumes of data, it is particularly important that experiments are able to quickly develop methods for separating signal and background. Using the $^{46}\text{Ar}(p, p)$ experiment, which directed a beam of ^{46}Ar ions into the AT-TPC filled with isobutane, a study was performed to determine if CNNs could improve the selection of resonant proton scattering events [42].

In neutrino and collider experiments previously mentioned, the typically technique is to train a CNN based on a leading architecture using a training sample composed of high-quality simulated data.

Since the available simulations capture many of the important features expected in data, it is typically assumed that this selector will generalize well to experimental data. This is not a good assumption for the experiments which are conducted at the AT-TPC. For instance, in the $^{46}\text{Ar}(p,p)$ experiment, the signal (proton) and one common background (carbon atoms) can be accurately simulated; however, all other backgrounds cannot be. Therefore, CNNs trained with either simulated data, or a small amount of hand-labeled experimental data were studied. Figure 11 shows examples of simulated and real data for proton, carbon, and other categories.

In total, 28,000 simulated training images were produced for each category. Since manually classifying data is time consuming, only 663 proton, 340 carbon, and 1686 other real training images were produced. These training sets are too small to be successful using deep CNNs, so they explored the use of transfer learning. This technique uses the fact that the feature extractor portion of the network is

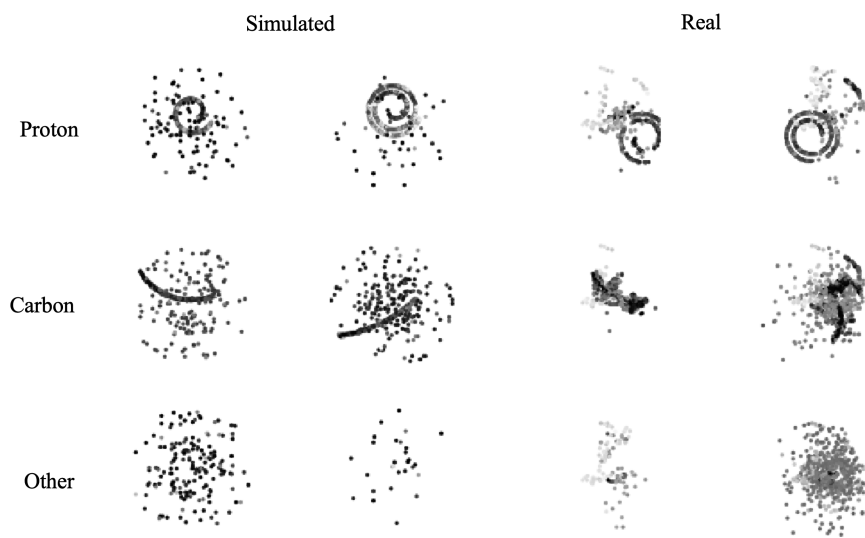


Fig. 11. Examples of simulated and real training images for the $^{46}\text{Ar}(p,p)$ experiment, where the real training images were extracted through manual classification. The images represent projections of the three-dimensional event data onto the xy -plane. Figure reproduced from [42].

designed to extract low-level features that are properties of images themselves rather than the particular dataset they were trained on while the classifier portion of the network is more tightly tied to the detail of the problem being solved [43].

Therefore, a network trained on one dataset may be usable for another dataset after applying a fine tuning procedure. In this procedure, the classifier portion of the network is removed and replaced with a new one with the correct number of inputs, and the network is retrained using a low learning rate. In the simplest version of this training, only the weights in the classifier portion are allowed to change. If the new problem is sufficiently different, it may be necessary to also allow the weights in the feature extractor to change as well.

To test transfer learning with $^{46}\text{Ar}(p, p)$ data, a VGG network [44] previously trained using ImageNet data [45] was fine tuned using either simulated or real data. The success of the training was judged using the F1 metric which can be written as

$$F1 = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \quad (2)$$

where precision is the fraction of true positives out of all positive selected events and recall is the fraction of true positives out of all true events. In particle physics contexts, precision is often referred to as purity and recall is referred to as efficiency. When a network that was fine tuned using simulated data was tested on simulated data, the F1 score was 1.0, signifying perfect classification. However, when the same network was tested with real data, the F1 score dropped to 0.67. This large drop in classification ability is directly related to the low fidelity of the simulated sample. When a network fine tuned on real data was tested on real data, the F1 score recovered to 0.93.

This result has a number of interesting implications. First, the VGG network that was used had been trained on ImageNet data which consists of natural images found on the internet. Natural images are very different from physics data in that they tend to be information dense while physics data is usually very sparse. Therefore, it is surprising that transfer learning works at all. Second, using

an exceptionally small sample of manually classified real data (only 663 proton signal examples), it was possible to obtain a selector of similar quality when tested with real data as one trained on simulated data and tested on simulated data. While neutrino and collider experiments have higher quality simulations than those available for the $^{46}\text{Ar}(p, p)$ experiment, there are still concerns about CNNs trained on simulated data increasing the systematic uncertainties of an experiment due to being the network learning the details of the model used in the simulation. These transfer learning results show that it may be possible to insulate a CNN from such model biases using small quantities of manually classified data.

5. Convolutional Neural Networks in Heterogeneous Collider Detectors

Unlike neutrino experiments and other experiments using TPC technology, detectors placed around the collision points of accelerators typically have a cylindrical geometry which the axis of the cylinder aligned with the colliding beams. Detectors like CMS [46] and ATLAS [47], located at the Large Hadron Collider are composed of many heterogeneous detector systems organized in concentric layers around the beam axis. The innermost detectors are usually tracking chambers, consisting of either drift chambers or silicon detectors, designed to measure the trajectory and momentum of charged particles. Placed at larger radii are the electromagnetic calorimeter (ECAL) and hadronic calorimeter (HCAL) systems designed to measure the energy deposited by a variety of particle types. The outermost layer is designed to identify muons which tend to penetrate much farther than other charged particles. Furthermore, while these detectors are azimuthally symmetric, they have a projective geometry such that detector components are smallest transverse to the beam (at low pseudorapidity) and are largest in very forward regions (at high pseudorapidity).

Due to these characteristic features, it is more challenging to interpret nearly raw collider data as images. For any given concentric layer, a popular choice is to unroll the layer at a chosen value of the

azimuthal angle so that individual pixels represent bins of pseudorapidity η and azimuthal angle ϕ .

Quark and gluon jet discrimination: A study of quark and gluon jet discrimination using CMS Open Data was presented in [48]. Figure 12 shows how the three images for each event (p_T weighted positions on the front face of the ECAL, and the energy deposits in the ECAL and HCAL, respectively) are produced from the CMS detector geometry. Each of the images are produced using the (η, ϕ) coordinate system, with the same binning scheme used for the two ECAL images, and the HCAL has five times coarser images. In order to distinguish between jets initiated by quarks or gluons, an algorithm based on ResNet-15 [35, 36] was developed that operates on the three images. The results were compared to more traditional techniques using summary information of the reconstructed jets and the CNN approach outperformed them all, achieving an ROC AUC value of 0.8077 ± 0.0003 compared to 0.8017 ± 0.0003 for the best of the other algorithms.

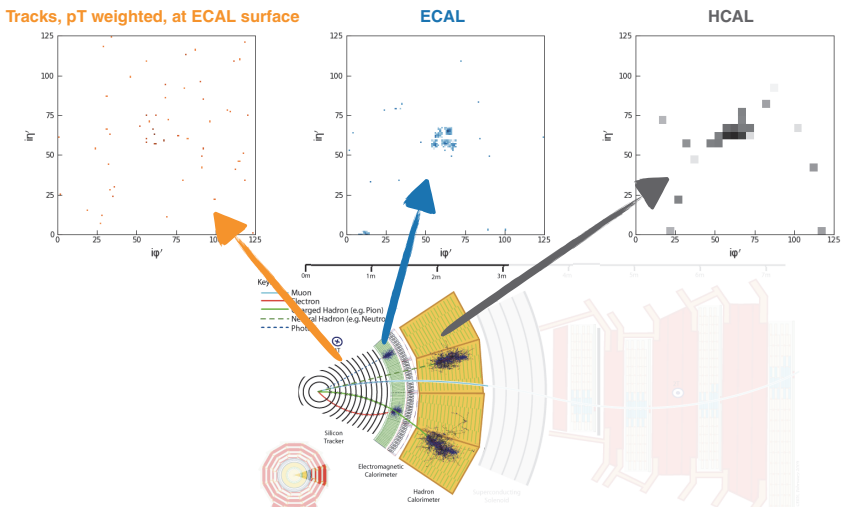


Fig. 12. An illustration of the CMS geometry and how to summarize information from the tracking system and the electromagnetic and hadronic calorimeters. Each system has a barrel-shaped geometry which must be recast to form 2D images. Figure reproduced from [48].

Electromagnetic shower particle identification: Derived variables based on charge and position that describe the shape of energy deposits in collider experiment calorimeters (also known as showers) have traditionally been used to classify showers initiated by different types of particles. In order to distinguish between electron, photon and charged pion showers, the algorithms described in [49] aim to go beyond these variables and use raw data images from the calorimeters.

A six-layer MLP neural network operating on 20 shower shape variables provides the baseline algorithm. These 20 variables summarize the information encoded in the raw detector data. Four other networks are considered, the first of which uses the same architecture operating on the 504 calorimeter pixels instead of the 20 variables. Three other networks that operate on three images, one from each layer of the calorimeter, have CNN-based architectures: the locally connected network (LCN) [50], a similar network with the LCN layers replaced by standard 2D convolutions, and a network based on DenseNet [51].

Figure 13 shows a comparison of the performance of the five networks for the task of electron–photon separation (left) and

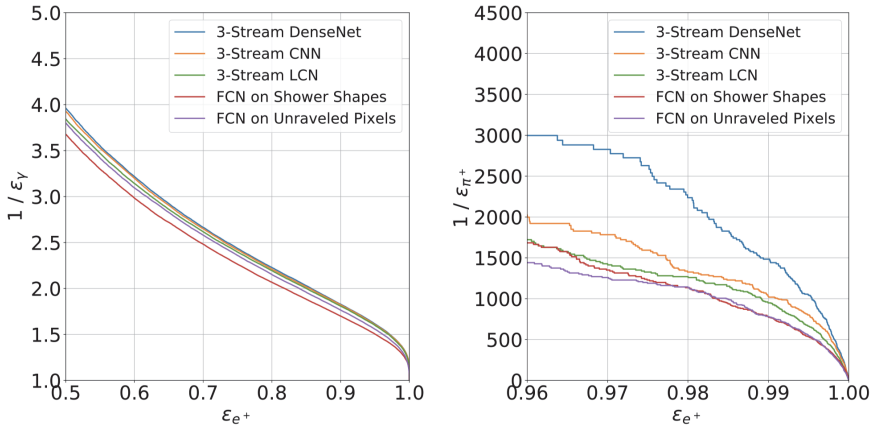


Fig. 13. A comparison of five different algorithms in the task of electron-photon (left) and electron-pion (right) shower discrimination. Figure reproduced from [49].

electron–pion separation (right). The three CNN-based algorithms outperform the MLP-based ones for both tasks, and the DenseNet-based algorithm demonstrates the best performance overall. It clearly shows that a considerable amount of information is lost in the construction of the shower shape variables and that this extra information is leveraged by the CNNs to significantly improve the performance.

6. End-to-End Analysis of Time Series Using One-Dimensional CNNs

The data produced by a single sensor is generally a continuous waveform: a signal that varies as a function of time. In many cases this one-dimensional (1D) representation of data needs to be processed without combining data from multiple sensors to form images, for example, when there is a need for pre-processing of the data from each individual sensor, or when the number of sensors is small and each will be processed individually.

A natural way to process these 1D waveforms is to use 1D convolutions. The n -element filters are applied to the input waveform to extract features in an analogous way to the extraction of image features in the 2D case as previously discussed. For example, a 1D convolution algorithm could be used to find peaks in a waveform to find energy deposits recorded by a given sensor. Other neural networks, such as recurrent neural networks (RNNs) [52, 53] and their subclass long short-term memory networks (LSTMs) [54], can be used on 1D waveforms, but 1D CNNs work very well on fixed length inputs such as those from detector elements with a fixed-length read-out window. An example of event classification using a 1D CNN is given below.

Pulse Shape Discrimination for Scintillation Signals: Pulse shape discrimination, the ability to identify different signals in raw waveforms, is a common task in high-energy physics. In this example [55], the experimental setup consists of a $^6\text{LiF:ZnS(Ag)}$ phosphor screen coupled to a scintillator cube, technology similar to that used

in the SoLiD experiment [56]. The light produced inside the scintillator cube was read out using a photomultiplier tube (PMT) and two silicon photomultipliers (SiPM). The scintillator cube was sensitive to interactions from gamma-rays and electrons that produce scintillation light signals referred to as electron scintillation (ES). The phosphor screen was sensitive to nuclear interactions producing a different light signal, referred to as nuclear scintillation (NS). The goal of the experiment was to distinguish between the ES and NS events from the raw SiPM waveforms, where Fig. 14 shows the average NS (top) and ES (bottom) waveforms. The PMT served two purposes: to trigger the readout of the SiPM waveforms, and to label the waveforms, with approximately 99% accuracy, as either an ES

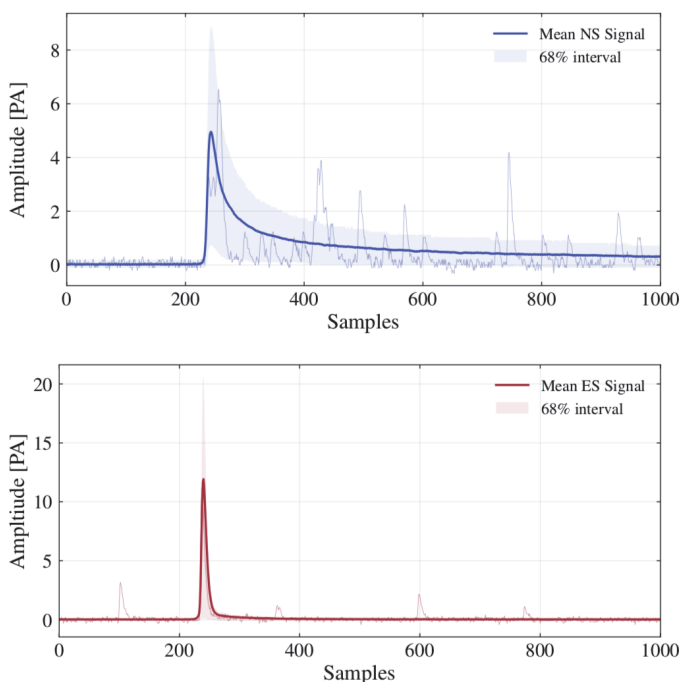


Fig. 14. The average waveform for NS (top) and ES (bottom) events where the shaded regions show the 68% interval of the ensemble used to calculate the average. Also shown in both panels is an example waveform. Figure reproduced from [55].

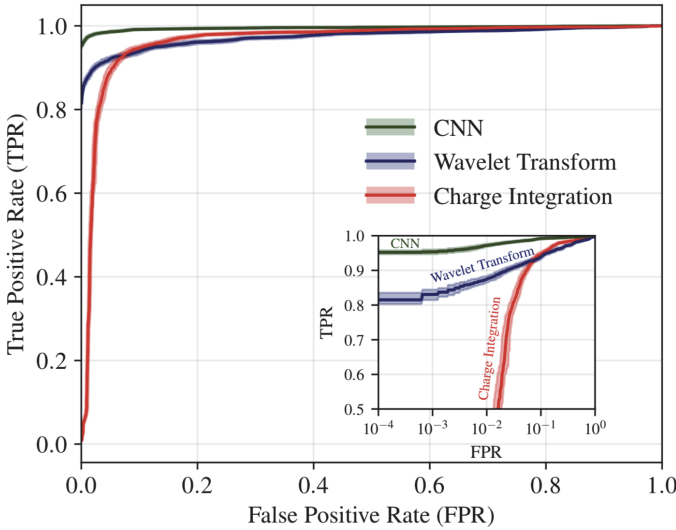


Fig. 15. ROCs curves for the different PSD algorithms. Figure reproduced from [55].

or NS event to avoid the need for simulations. Each of the SiPM waveforms consisted of 1000 samples.

A 1D CNN was developed using just two convolutional layers, each followed by a max pooling layer. The output of the second pooling layer fed into a fully connected layer, and finally a single output node with a softmax activation to provide the probability of the waveform being of the NS type. The CNN algorithm significantly outperforms two more traditional approaches, as demonstrated by the distributions in Fig. 15.

7. Graph Neural Networks for Large Three-Dimensional Detectors

Graph neural networks are a more recent development than CNNs, and as such GNNs are currently less commonly used in high-energy physics than CNNs. However, GNNs are beginning to be used in event reconstruction [57–59], as described in detail in Chapter 12. It is likely only a matter of time before many examples of end-to-end

analysis using GNNs become apparent, but the only current example is discussed below.

Event classification in the IceCube experiment: IceCube [60] is an experiment located in Antarctica that aims to measure interactions of atmospheric and astrophysical neutrinos. The detector consists of a series of photomultiplier tube detector modules (called DOMs) buried in the ice to measure Cherenkov radiation produced by charged particles traveling in the ice. There are approximately 6000 DOMs arranged in an irregular 3D hexagonal geometry making it well-suited to graph representation. Each of the DOMs is represented as a graph node with six features: the (x, y, z) position, the sum of the charge in the first detected pulse, the sum of the charges from all pulses, and the time at which the first pulse went above threshold. On an event-by-event basis, only those DOMs that record a signal are added as nodes to the graph. The goal of the GNN is to classify an event (i.e. the graph as a whole) as either a signal neutrino interaction or a background event in an environment where the signal interactions are very rare compared to the backgrounds. The network architecture is based on the MoNet model [61].

The local neighborhood of each node, or its adjacency to other nodes, is defined using the three spatial position features (x, y, z) . The edges between nodes are assigned weights from a Gaussian distribution that depends on the distance between the two nodes. The width of this Gaussian distribution is a learned network parameter that controls how quickly information is spread between spatially distant nodes. A series of convolutions are applied to the graph followed by a logistic regression to predict the event type. Figure 16 shows the true positive rate as a function of the false positive rate and demonstrates that the GNN significantly outperforms traditional methods as well as 3D CNN approaches, achieving a signal-to-noise ratio of 2.98 compared to the baseline of 0.987 [62].

8. Opening the Black-Box

In traditional selection techniques, reconstructed features are designed to quantify a physical property known to differ between

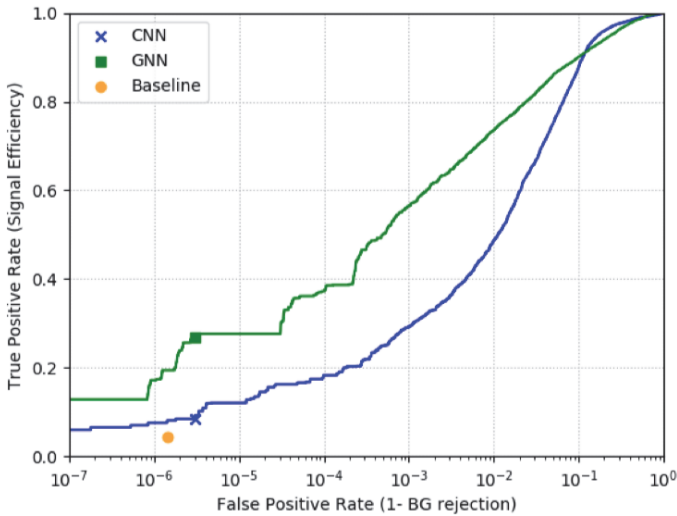


Fig. 16. Distribution showing the signal efficiency as a function of the false positive rate. The IceCube GNN algorithm (green) is compared to a 3D CNN (blue) and a baseline point (yellow) from traditional techniques. Figure reproduced from [62].

event types. For instance, the multiple scattering distribution for muons and charged pions differ because muons only scatter due to the Coulomb potential of the material they are propagating through while charged pions also scatter due to the strong nuclear potential. Therefore, it is reasonable to expect that summary statistics like the mean and RMS of the scattering angles of a track may be useful when classifying tracks as having been created by a muon or charged pion.

Since end-to-end approaches use more information than these summary statistics, they typically perform better, but it is more difficult to attribute their performance to understandable physical properties. Moreover, using more information potentially exposes the algorithm to learning spurious or incorrect details due to imperfections in the simulated training dataset. Therefore, it is critical to have tools for interrogating the network to determine on what basis it is making its decisions. For a final physics analysis, this involves a black-box input-output analysis where systematically-varied simulation samples are classified by the network to determine how sensitive

the classification is to plausible variations in the dataset. However, tools which provide a qualitative understanding the network's decisions can provide additional assurance of reasonableness. In this section, we will discuss examining feature maps at various depths in the network to identify features frequently associated with certain classifications, low-dimensional visualizations of the features produced by the final layer of the network to determine how a test sample forms clusters, and occlusion tests to determine what parts of an image are more salient for making a decision. In all of the cases given below, the methods produce figures that are inspected by eye to determine if the behavior appears reasonable; there is no unique quantitative figure-of-merit that works for all networks.

8.1. *Feature maps*

The outputs of the convolutional layers can provide insight as to what features are being extracted from the input images. Looking at the output of the first convolutional layer for different types of events can show what sort of features the network is looking for in order to classify the event. However, the layer outputs become increasingly abstract as the depth into the network increases making visual inspection of the deeper layer outputs difficult.

Figure 17 shows example feature maps from the 1D CNN described in [55] and Sec. 6, for the first convolutional (left) and second (right) convolutional layers for the two types of signals. For example, visual inspection of the distributions on the left shows that filter six responds most to low amplitude samples and filters 2 and 3 find the main peaks in the waveforms.

As an example from a 2D CNN, Figure 18 shows the response from the first and final layers of the DUNE CVN [9] for an input CC $\bar{\nu}_e$ interaction. The first convolutional layer consists of 64 learned 7×7 pixel filters, visualized in the top middle panel. The results of applying these 64 filters to the input image are shown on in the top right panel, demonstrating that some filters result in a weak (yellow) response and some give a strong (red) response for the chosen input image. Some of the filters can be seen to respond strongly to the

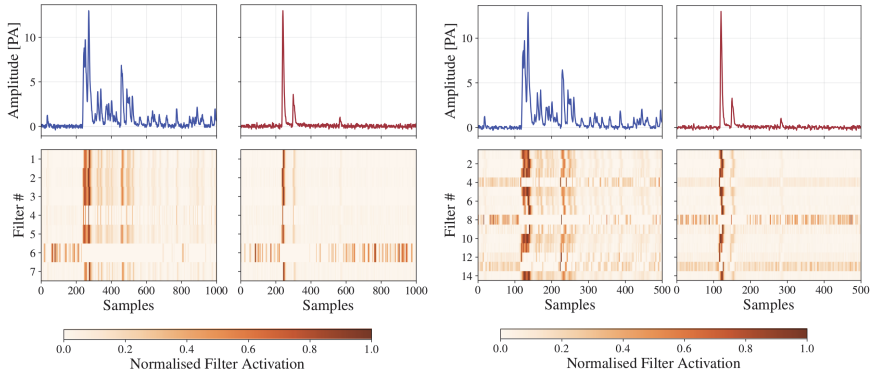


Fig. 17. Normalized feature outputs of the first (left) and second (right) convolutional layers for the two types of signal waveforms. The input signals on the right have been downsampled (to mimic the max pooling in the CNN architecture) to show the correlation with the second convolutional layer. Figure reproduced from [55].

central part of the shower, others to the sparser halo pixels, and some have a weak response for the whole image. The 512 feature maps from the final convolutional layer, shown in the bottom panel, have a large variety in response but are very abstracted since the many pooling layers have downsampled the original 500×500 pixel input down to 16×16 pixels in the feature maps.

8.2. Low-dimensional visualizations

For both CNNs and GNNs discussed in this chapter, the network consists of a feature-extractor and classification sub-network trained simultaneously. In the CNN case, the feature extractor consists of a stack of convolutional layers, with the possibility of pooling or other types of layers. In the GNN case, the feature extractor typically involved convolutions generalized to the graph structure and message passing steps. In either case, the feature extractor produces a high-dimensional vector encoding of the information contained within the input data. The classification sub-network usually consists of a single fully connected layer with as many output nodes as classes and the outputs recast as class probabilities using the softmax function.

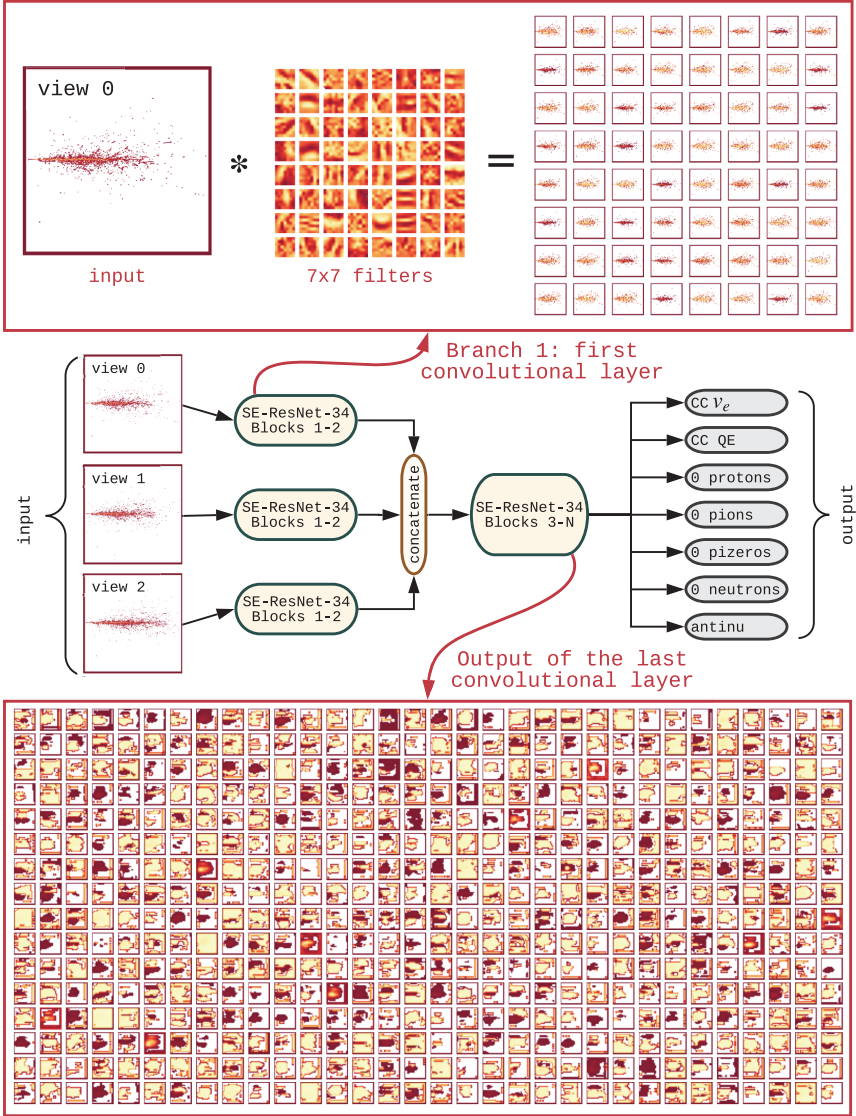


Fig. 18. The outputs from the first (top right) and last (bottom) convolutional layers in the DUNE CVN for an input CC $\bar{\nu}_e$ interaction. The response strength is shown ranging from yellow for low response values to red for a strong response, and white is used for no response due to empty input pixels. Figure reproduced from [9].

In the simplest case, predicted outputs are merely rescaled linear combinations of the output of the feature extractor making decision boundaries in the high-dimensional feature space encoding hyperplanes. Therefore, in a successfully trained network, examples from the same class should be separated by a small Euclidean distance, while examples from different classes should be separated by a large Euclidean distance.

Examining the shape of clusters within the high-dimensional feature space could provide insights into why different examples are classified correctly or incorrectly. For instance, two classes may exist as two well-separated clusters except at one surface where they touch. At the point of contact, the proper classification of those examples would be ambiguous. This makes it possible to isolate only those examples with a true ambiguity. Unfortunately, the output of the feature extractor typically has a dimensionality of $\mathcal{O}(1000)$, well beyond the bounds of normal visualization techniques.

The t -distributed stochastic neighbor embedding (t -SNE) technique [63] provides a method to visualize high-dimensional data by embedding it in a lower dimensional space. To do this, the similarity representing the probability that two points are neighbors is constructed for each pair of points in either the high- or low-dimensional space as a function of the Euclidean distance between the points. In the high-dimensional space, the similarity is based on a Gaussian probability density while it is based on Student's t -distribution in the low-dimensional space. The Student's t -distribution prevents points from crowding in the low-dimensional space. Finally, the points in the low-dimensional space are rearranged to minimize the Kullback–Leibler divergence between the similarity distributions in both representations assuring that points have the same relationship to their neighbors in low dimensions as in high.

Figure 19 shows the t -SNE algorithm [63] applied to 1024-dimensional outputs of the NOvA CNN [8]. The structure of the clusters gives insight into what features the network primarily used to separate neutrino interactions. Looking at visualizations of a representative sample of interactions, it is clear what characteristics define the axes of the low-dimensional representation. On the horizontal



Fig. 19. A visualization of the t-SNE algorithm applied to the 1024 dimensional outputs of the NOvA CNN [8]. Figure courtesy of the NOvA collaboration.

axis, events become dominated by a long track on the right side and dominated by an electromagnetic shower on the left side. On the vertical axis, the multiplicity of objects in the interaction increases while moving from the bottom to the top of the figure.

With this scheme in hand, we can understand how the clusters of classes are arranged. We see that ν_μ are well separated from ν_e with neutral current interactions in the middle. This makes sense since ν_μ interactions are dominated by tracks while ν_e interactions are dominated by an electromagnetic shower. Neutral current interactions lack a charged lepton, but they can contain either a charged pion (track-like) or a neutral pion (shower-like) which naturally causes ambiguities with either ν_μ or ν_e . Cosmic rays largely occupy the lower right side of the figure since they are dominated by single muons. Finally, ν_τ interactions are clustered in the top half of the figure since they only occur at higher energies, and they are poorly separated from the three other neutrino channels since the τ -lepton can decay either hadronically or leptonically. In broad terms, this figure suggests that the network is relying on similar topological features to what a hand

scanner might use. Furthermore, it suggests potential improvements to the network. For instance, since cosmic rays consisting of single muons are easy to separate, and the selector may be improved by constructing a biased sample of cosmic rays emphasizing more complicated topologies which are rarer, but more likely to be mis-categorized as a neutrino. Similarly, maintaining a single category for ν_τ , regardless of the decay mode of the τ -lepton needlessly confuses the network.

8.3. Occlusion tests

While t -SNE provides general insight into what type of examples are seen as similar by the network, it can be useful to determine precisely which portions of an example are salient to the decision made by the network. A very simple method for determining salience is the occlusion test. In an occlusion test, a small portion of an input example is withheld from the network. In the CNN case, this would mean changing a small patch of pixels to zeros. The change in the network output is placed into a separate map at the pixel corresponding to the center of the occluded region in the input image. Repeating this across the image produces a salience map showing which regions were most important in making a particular decision.

Figure 20 shows the salience map for a deep inelastic scattering ν_e interaction using the NOvA CNN [8]. In this case, the occluded region consisted of a movable 5×5 square of pixels. Since the image shown corresponds to a deep inelastic scatter, the interaction consists of an electron produced by the charged current interaction and an array of charged and neutral pions produced by the struck nucleus. Despite the high multiplicity of this interaction, the only region which reduces the ν_e score of this interactions is near the start of the electron shower. This is consistent with how a hand scanner would classify this event since electrons produce a single track before they initiate a shower while photons are invisible until they initiate their first pair production. Furthermore, the lack of dependence on the details of the particles produced by the struck nucleus

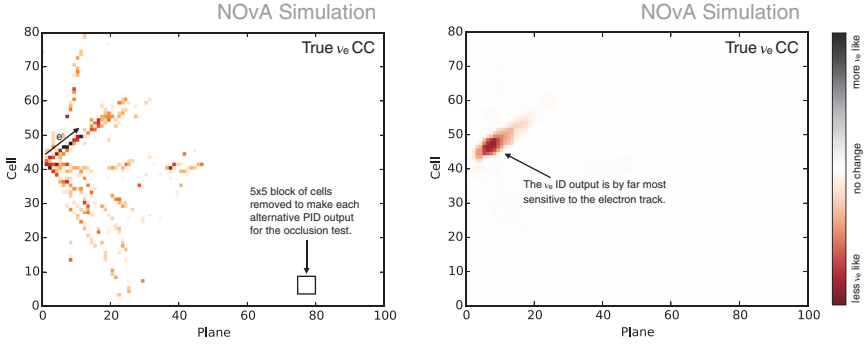


Fig. 20. An occlusion test demonstrating the most salient parts of an input image to the decision made by the NOvA CNN [8]. (Left) A single view of a true ν_e -CC interaction. This interaction consists of a single electromagnetic shower from a primary electron along with several track-like objects from the hadrons produced by the nucleus. (Right) The change in the ν_e -CC score as a function of the location of a 5×5 occluded region. Figures courtesy of the NOvA collaboration.

is reassuring since that portion of the simulation is typically less realistic.

9. Conclusions

End-to-end analyses using deep learning are becoming widespread in high-energy physics, taking raw detector data as input and providing physics-level outputs such as event classification. The majority of algorithms to date are based on 2D convolutional neural networks applied to images of the experimental detector data, but other approaches such as graph neural networks are gaining popularity. The examples presented here demonstrate that end-to-end deep learning analyses are very powerful because they have access to all of the detector information and they can significantly outperform more traditional analysis techniques.

As with any type of analysis, it is important to ensure robustness and to understand how the event classification is being performed. The techniques outlined in Sec. 8 help to elucidate how the deep neural networks extract features from the input data and use the

features to perform separation of the different categories within some high-dimensional space.

References

- [1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, Backpropagation applied to handwritten zip code recognition, *Neural Comput.* **1** (1989) 541.
- [2] G. Giacomelli, Introduction to the Workshop, in *30 Years of Bubble Chamber Physics* (2003); arXiv:physics/0604152.
- [3] MINOS Collaboration, D. Michael *et al.*, The magnetized steel and scintillator calorimeters of the MINOS experiment, *Nucl. Instrum. Meth. A* **596** (2008) 190; arXiv:0805.3170 [physics.ins-det].
- [4] NOvA Collaboration, D. Ayres *et al.*, The NOvA technical design report (2007).
- [5] DUNE Collaboration, B. Abi *et al.*, Deep Underground Neutrino Experiment (DUNE), far detector technical design report, Volume II DUNE physics (2020); arXiv:2002.03005 [hep-ex].
- [6] C. Rubbia, The liquid argon time projection chamber: A new concept for neutrino detectors (1977), CERN-EP-INT-77-8.
- [7] A. M. Holin, Electron neutrino appearance in the MINOS experiment, Ph.D. thesis, UCL (University College London) (2010).
- [8] A. Aurisano *et al.*, A convolutional neural network neutrino event classifier, *J. Instrum.* **11**(09) (2016) P09001.
- [9] DUNE Collaboration, B. Abi *et al.*, Neutrino interaction classification with a convolutional neural network in the dune far detector, *Phys. Rev. D* **102** (2020) 092003.
- [10] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* (Spartan Books, 1962).
- [11] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning* (MIT Press, 2016).
- [12] D. H. Hubel and T. N. Wiesel, Receptive fields of single neurones in the cat's striate cortex, *J. Physiol.* **148** (1959) 574.
- [13] D. H. Hubel and T. N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *J. Physiol.* **160** (1962) 106.
- [14] D. H. Hubel and T. N. Wiesel, Receptive fields and functional architecture of monkey striate cortex, *J. Physiol.* **195** (1968) 215.
- [15] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, *Commun. ACM* **60** (2017) 84.
- [16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, ImageNet large scale visual recognition challenge, *Int. J. Computer Vision* **115** (2015) 211.
- [17] A. Sperduti and A. Starita, Supervised neural networks for the classification of structures, *IEEE Trans. Neural Networks* **8** (1997) 714.

- [18] M. Gori, G. Monfardini and F. Scarselli, A new model for learning in graph domains, in *Proc. 2005 IEEE Int. Joint Conf. Neural Networks*, 2005, Vol. 2 (2005) p. 729.
- [19] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, The graph neural network model, *IEEE Trans. Neural Networks* **20** (2009) 61.
- [20] J. Bruna, W. Zaremba, A. Szlam and Y. LeCun, Spectral networks and locally connected networks on graphs (2013); arXiv:1312.6203.
- [21] M. Henaff, J. Bruna and Y. LeCun, Deep convolutional networks on graph-structured data (2015); arXiv:1506.05163.
- [22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S. Yu, A comprehensive survey on graph neural networks, *IEEE Trans. Neural Networks Learning Syst.* (2020) 1–21.
- [23] H. Robbins and S. Monro, A stochastic approximation method, *Ann. Math. Statist.* **22** (1951) 400.
- [24] M. D. Zeiler, Adadelta: An adaptive learning rate method (2012); arXiv:1212.5701.
- [25] G. Hinton, Neural networks for machine learning, http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.
- [26] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2017); arXiv:1412.6980.
- [27] S. Ruder, An overview of gradient descent optimization algorithms (2016); arXiv:1609.04747.
- [28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, in *2015 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2015).
- [29] M. Lin, Q. Chen and S. Yan, Network in network (2013); arXiv:1312.4400 (2014).
- [30] NOvA Collaboration, P. Adamson *et al.*, Constraints on oscillation parameters from ν_e appearance and ν_μ disappearance in NOvA, *Phys. Rev. Lett.* **118** (2017) 231801; arXiv:1703.03328 [hep-ex].
- [31] NOvA Collaboration, P. Adamson *et al.*, Search for active-sterile neutrino mixing using neutral-current interactions in NOvA, *Phys. Rev. D* **96** (2017) 072006; arXiv:1706.04592 [hep-ex].
- [32] MicroBooNE Collaboration, R. Acciarri *et al.*, Design and construction of the microBooNE detector, *J. Instrum.* **12** (2017) P02017; arXiv:1612.05824 [physics.ins-det].
- [33] MicroBooNE Collaboration, R. Acciarri *et al.*, Convolutional neural networks applied to neutrino events in a liquid argon time projection chamber, *J. Instrum.* **12** (2017) P03011; arXiv:1611.05531 [physics.ins-det].
- [34] S. Ren, K. He, R. Girshick and J. Sun, Faster R-CNN: Towards real-time object detection with region proposal networks, in *Proc. 28th Int. Conf. Neural Information Processing Systems, Volume 1, NIPS'15* (MIT Press, Cambridge, MA, 2015).
- [35] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition (2015); arXiv:1512.03385.

- [36] K. He, X. Zhang, S. Ren and J. Sun, Identity mappings in deep residual networks (2016); arXiv:1603.05027.
- [37] J. Hu, L. Shen and G. Sun, Squeeze-and-excitation networks (2017); arXiv:1709.01507.
- [38] DUNE Collaboration, B. Abi *et al.*, Long-baseline neutrino oscillation physics potential of the DUNE experiment, *Eur. Phys. J. C* **80** (2020) 978.
- [39] M. Auger *et al.*, The EXO-200 detector, part I: Detector design and construction, *J. Instrum.* **7** (2012) P05010; arXiv:1202.2192 [physics.ins-det].
- [40] EXO Collaboration, S. Delaquis *et al.*, Deep neural networks for energy and position reconstruction in EXO-200, *J. Instrum.* **13** (2018) P08023; arXiv:1804.09641 [physics.ins-det].
- [41] J. Bradt *et al.*, Commissioning of the active-target time projection chamber, *Nucl. Instrum. Meth. A* **875** (2017) 65.
- [42] M. P. Kuchera, R. Ramanujan, J. Z. Taylor, R. R. Strauss, D. Bazin, J. Bradt and R. Chen, Machine learning methods for track classification in the AT-TPC, *Nucl. Instrum. Meth. A* **940** (2019) 156; arXiv:1810.10350 [cs.CV].
- [43] J. Yosinski, J. Clune, Y. Bengio and H. Lipson, How transferable are features in deep neural networks?, in *Proc. 27th Int. Conference on Neural Information Processing Systems, Vol. 2, NIPS'14* (MIT Press, Cambridge, MA, 2014).
- [44] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, in *International Conference on Learning Representations* (2015).
- [45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-scale hierarchical image database, in *CVPR09* (2009).
- [46] CMS Collaboration, S. Chatrchyan *et al.*, The CMS Experiment at the CERN LHC, *J. Instrum.* **3** (2008) S08004.
- [47] ATLAS Collaboration, G. Aad *et al.*, The ATLAS Experiment at the CERN large hadron collider, *J. Instrum.* **3** (2008) S08003.
- [48] J. Alison, S. An, P. Bryant, B. Burkle, S. Gleyzer, M. Narain, M. Paulini, B. Poczos and E. Usai, End-to-end particle and event identification at the large hadron collider with CMS open data, in *Meeting of the Division of Particles and Fields of the American Physical Society (DPF2019)*, Boston, MA, July 29–August 2, 2019 (2019); arXiv:1910.07029 [hep-ex].
- [49] L. De Oliveira, B. Nachman and M. Paganini, Electromagnetic showers beyond shower shapes, *Nucl. Instrum. Meth. A* **951** (2020) 162879; arXiv:1806.05667 [hep-ex].
- [50] L. de Oliveira, M. Paganini and B. Nachman, Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis, *Comput. Softw. Big Sci.* **1** (2017) 4; arXiv:1701.05927 [stat.ML].
- [51] G. Huang, Z. Liu and K. Q. Weinberger, Densely connected convolutional networks (2016); arXiv:1608.06993.
- [52] W. A. Little, The existence of persistent states in the brain, *Math. Biosci.* **19** (1974) 101.

- [53] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci.* **79** (1982) 2554.
- [54] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.* **9** (1997) 1735–1780.
- [55] J. Griffiths, S. Kleinegesse, D. Saunders, R. Taylor and A. Vacheret, Pulse shape discrimination and exploration of scintillation signals using convolutional neural networks, preprint (2018); arXiv:1807.06853 [physics.ins-det].
- [56] SoLid Collaboration, Y. Abreu *et al.*, A novel segmented-scintillator antineutrino detector, *J. Instrum.* **12** (2017) P04024; arXiv:1703.01683 [physics.ins-det].
- [57] S. Farrell, P. Calafiura, M. Mudigonda, Prabhat, D. Anderson, J.-R. Vlimant, S. Zheng, J. Bendavid, M. Spiropulu, G. Cerati, L. Gray, J. Kowalkowski, P. Spentzouris and A. Tsaris, Novel deep learning methods for track reconstruction (2018); arXiv:1810.06111.
- [58] S. R. Qasim, J. Kieseler, Y. Iiyama and M. Pierini, Learning representations of irregular particle-detector geometry with distance-weighted graph networks, *Eur. Phys. J. C* **79** (2019).
- [59] X. Ju, S. Farrell, P. Calafiura, D. Murnane, Prabhat, L. Gray, T. Klijnsma, K. Pedro, G. Cerati, J. Kowalkowski, G. Perdue, P. Spentzouris, N. Tran, J.-R. Vlimant, A. Zlokapa, J. Pata, M. Spiropulu, S. An, A. Aurisano, J. Hewes, A. Tsaris, K. Terao and T. Usher, Graph neural networks for particle reconstruction in high energy physics detectors (2020); arXiv:2003.11603.
- [60] IceCube Collaboration, M. Aartsen *et al.*, The IceCube Neutrino Observatory: Instrumentation and Online Systems, *J. Instrum.* **12** (2017) P03012; arXiv:1612.05093 [astro-ph.IM].
- [61] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda and M. M. Bronstein, Geometric deep learning on graphs and manifolds using mixture model CNNs (2016); arXiv:1611.08402.
- [62] N. Choma, F. Monti, L. Gerhardt, T. Palczewski, Z. Ronaghi, M. Prabhat, W. Bhimji, M. Bronstein, S. Klein and J. Bruna, Graph neural networks for IceCube signal classification (2018); arXiv:1809.06166.
- [63] L. van der Maaten and G. Hinton, Visualizing data using *t*-SNE, *J. Mach. Learn. Res.* **9** (2008) 2579.

This page intentionally left blank

Chapter 11

Clustering

Kazuhiro Terao

*SLAC National Accelerator Laboratory, 2575 Sand Hill Rd.,
Menlo Park, CA 94025, USA
kterao@slac.stanford.edu*

Clustering methods are in the core of data reconstruction in particle physics. Recent advancements in machine learning and computer vision offer a number of new, powerful techniques to be explored including image segmentation techniques with convolutional neural networks and a generalization of clustering tasks using graph neural networks. Applications of modern machine learning techniques for clustering tasks in pipelines of physics data reconstruction are discussed.

1. Introduction

Clustering in data analysis refers to partitioning of data using common features. In this manuscript, applications of *clustering methods for physics reconstruction in particle physics data*, that utilize machine learning (ML) techniques, are described. Reconstruction is a process of inferring physics phenomena that took place in an experiment's detector and recorded in the data produced by the detector. Particle physics detectors consist of many sensors, even up to millions, in order to capture the full details of a particle interaction. Signals are correlated across many sensors and may be interpreted as sets, for example, photo-electrons detected across photo-multiplier tubes (PMTs) in the Super-Kamiokande detector collectively represent a Cherenkov ring for a charged particle, and charge collected in the pixels of liquid argon time projection chambers (LArTPCs) represent individual particle trajectories.

Clustering methods are at the core of reconstruction process and are required for identifying individual particle instances in particle imaging detectors. Similarly, they are used for the identification of a subset of particles that originate from the same high-energy collision, which helps to disentangle an individual interaction of interest from the many simultaneous, low-energy *pile-up* interactions that are overlaid. Much progress has been made in clustering techniques using deep neural networks (DNNs) in the field of computer vision [1–4], and the development of scientific applications in particle physics has been explored in depth within the community of large particle imaging detectors. This review chapter summarizes these recent developments to provide a summary knowledge.

1.1. *Traditional clustering techniques*

While this section is not intended to be a comprehensive review of clustering techniques in data analysis, it is worth noting a few popular methods that are used in combination with ML techniques, to be introduced later, including DBSCAN and **k-means**. For a comprehensive review, readers are referred to introductory reviews and popular scientific software libraries such as **scikit-learn** [5] for insights and hands-on practice

It is important to note that, whether a method is supervised or unsupervised, the quality of the output of clustering algorithms is subjective as it requires some domain-specific knowledge or assumptions such as the measure of distance, density, or an underlying structure of hierarchy. While the output of clustering algorithms is useful for high-level interpretation and for finding the underlying nature of the data, it is not an objective piece of evidence for discovery.

1.1.1. *DBSCAN*

Density-based spatial clustering application with noise (DBSCAN) [6] is one of the most well known and frequently used clustering methods [7, 8]. Given a set of data points, the algorithm identifies *core points* and *outliers*, or noise, and forms clusters from the former. A point is a core point if there exist at least K points within a

distance ϵ where K specifies the minimum desired size of a cluster. The core points that are within the radius ϵ of each other as well as the other points that may lie within the distance ϵ are considered to belong to the same cluster. Intuitively, DBSCAN clusters points that are densely connected with a distance metric that may depend on the application.

One of drawbacks of using DBSCAN is the fact that ϵ can take only one value where, in general, one may expect different kinds of clusters with different density profiles. OPTICS [9] was introduced as an extension to DBSCAN where ϵ can take a form of a range rather than a particular value, and more recently HDBSCAN [10] was introduced as a density-based *hierarchical clustering* method. HDBSCAN does not require the ϵ hyperparameter. Instead, K is the only hyperparameter. Another drawback is its scalability to high-dimensional data, which ultimately depends on the distance definition. One typical distance metric is to use the Euclidean distance between points. However, as the dimension of space (i.e. number of features to describe each data instance) increases, the distance between two instances also becomes larger whether they belong to the same true underlying cluster or not. This is a challenge known as a “curse of dimensionality”. The same applies to any clustering method that utilizes the Euclidean distance.

1.1.2. *k-means clustering*

The *k-means* [11] predicts partitions given the number of clusters, k , as a hyperparameter. It first assigns all data points randomly over k clusters, and then computes the *cluster centroids*, which are the mean values of all points that belong to a given cluster. The calculation of this mean value is application specific (e.g. a geometrical mean position, the mean of values carried by each data point). Then points are reassigned to a cluster that has its mean value closest to a subject point. These steps are repeated until no more reassignment takes place. The *k-means* can also be considered as a particular implementation of the *expectation maximization* algorithm [12] for Gaussian mixture models.

There are two major drawbacks of k -means clustering. The first is that the performance is known to heavily depend on random cluster assignments at the initialization. Possible mitigations include sampling multiple initialization positions by simply running the algorithm multiple times, or a smarter initialization. As an example for the latter it may be preferable to have cluster centroids to be well spread and avoid having centroids that are close to each other at the initialization stage. Such an extension is implemented in the k -means++ algorithm [13].

The second is the necessity of knowing the number of clusters, k . A typical mitigation method is to scan different k values and compute the maximum cluster spread, which is a measure of the deviation of the data points within a cluster from the centroid. The spread may be large for a smaller k value than the optimal one, and the spread may suddenly drop when increasing k value to the correct number of clusters, which can be used as an estimation method for k . This is only an approximation that depends on the inter-cluster separation (e.g. the distance between centroids of true underlying clusters) and the intrinsic spread within each cluster.

1.2. *ML for clustering in data reconstruction*

Shallow neural networks (NNs) such as multi-layer-perceptrons (MLPs) have been used for supporting the clustering tasks in particle tracking reconstruction in experiments. For reconstructing particle tracks from detector hits in the ATLAS experiment [14], NNs have been used to identify merged clusters (i.e. trajectories) so that they can be recovered in an iterative process [15, 16]. In the LHCb experiment [17], NNs are used for checking the validity of a reconstructed track, which yields factor of two lower false-positive (i.e. background) trajectories compared to chi-square-based approach [18]. Traditionally, a major use of ML for clustering in the particle physics data reconstruction pipeline has been these shallow NNs to support the existing particle tracking algorithms.

More recently, deep-learning-based approaches [19–25] have been introduced for reconstructing particle trajectories and higher-level

reconstruction objects for large-scale LArTPC detectors in Micro-BooNE [26], ICARUS [27], and the Deep Underground Neutrino Experiment (DUNE) [28]. These methods are categorized by different objectives and techniques employed in the following review. While the characteristics vary quite a bit, all methods belong to a loose definition of clustering for identifying data partitions, and also originate from the area of geometric deep learning and computer vision, in particular the subfield called image segmentation.

2. Fixed Number of Partitions

Semantic segmentation is an image segmentation technique used to identify the type of an object that individual pixel represents. It casts a challenge of object classification down to the pixel level. Figure 1, taken from the PILArNet public data repository [29], shows an example of a semantic segmentation task. A semantic segmentation partitions data, image pixels, among the predefined set of categories (i.e. *semantics*). Two pioneering DNNs are the fully convolutional network (FCN) [30] and the U-Net [31]. Both of these belong to a family of convolutional neural networks (CNNs) [32–34], which have

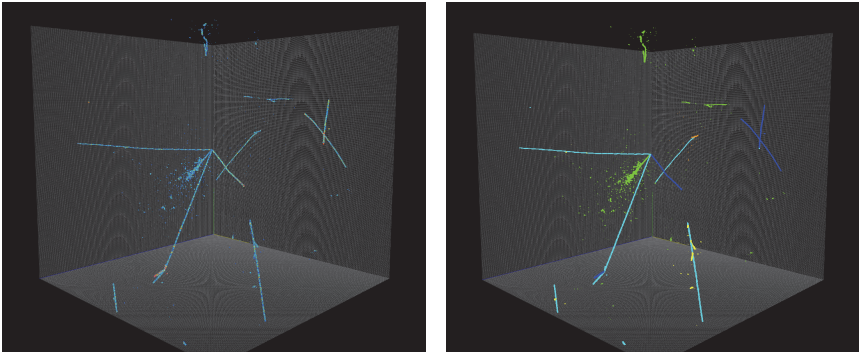


Fig. 1. Example of a semantic segmentation task. The left shows the input image with particle trajectories where the continuous color scale corresponds to the amount of energy deposited by a particle in each pixel. The right shows different semantics (i.e. particle types) in a discrete set of colors. This figure is taken from [29].

been extremely successful in the field of computer vision and established as the de-facto algorithm for image analysis.

2.1. *Deep neural networks for semantic segmentation*

CNNs were first applied to and found to be successful [34–37] for an image classification tasks, namely assigning an image to one of pre-defined set of categories (e.g. a cat vs. dog). A typical CNN for image classification consists of repeating blocks of convolution layers and pooling layers or strided convolution layers to downsample an input image in order to extract translationally invariant features at different spatial resolutions. This process gradually reduces the spatial size of an input data tensor and expands in the *feature* dimension, referred to as *channels* in image data. We refer to this as an *encoder architecture*, or simply an *encoder*, as what it does is to extract image features and encode into the feature dimension. At the end of an encoder is a block of fully-connected layers, which discards spatial information from the input data, to classify the whole image into one of several categories. Prior to this final block, a downsampled intermediate data tensor preserves information related to feature locations.

The FCN [30] explores the idea of reusing the spatial information in the output of a CNN encoder by replacing the last block of fully-connected layers with a 1×1 convolution layer. Intuitively this 1×1 convolution performs a semantic type classification at the downsampled, coarse pixel level. The goal of semantic segmentation is, however, to perform this classification at the spatial resolution of an input image. In order to do this, The FCN adds blocks of upsampling and convolutional layers. These blocks are essentially learnable interpolation algorithms. This process of transforming the encoded feature information back to a tensor of a high-spatial resolution is referred to as a *decoding* architecture, or simply a *decoder*. The very last layer consists of $N+1$ filters where N is the number of categories, or partitions, and one additional category represents the pixels that do not belong to any semantic type as background.

The U-Net [31] extends the idea of the FCN, which, despite being a successful application of CNNs for semantic segmentation task,

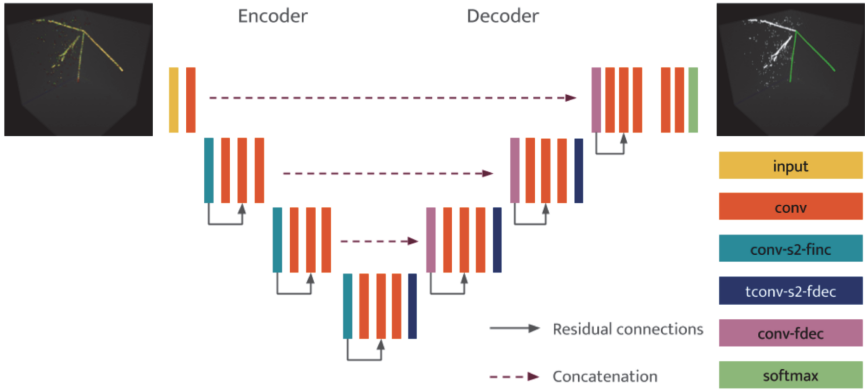


Fig. 2. Example implementation of a U-Net architecture for LArTPC neutrino detectors. The skip connections are shown in dashed arrows. This figure is taken from [23].

had limited spatial precision. The challenging aspect of FCN is that a full recovery of the spatial precision by an upsampling operation alone is impossible because the spatial information is evidently lost at the previous downsampling operations. In other words, the lost spatial information is present prior to each downsampling operation within the encoding blocks. U-Net exploited this fact by introducing *skip connections*, which is an operation to take a data tensor of corresponding size from the encoder and concatenate to the same, upsampled data tensor in the decoder. An example architecture diagram is shown in Fig. 2, taken from a U-Net implementation for a 3D imaging LArTPC detector [23]. With the skip connections, the lost spatial information is available to the convolution layers in the decoder, and results in a dramatic improvement in predicting the boundary of objects at the pixel level [31].

2.2. Semantic segmentation in neutrino detectors

One of the first applications of semantic segmentation in particle physics was applied by the MicroBooNE experiment [22] using U-Net architecture with residual connections [37], called U-ResNet, similar to the architecture shown in Fig. 2. The goal of their application was to partition image pixels into two types: pixels that belong to a *track*,

a trajectory with a line-like topology, and the others that belong to a *shower*, a trajectory with many branches and scatterings. Due to the distinct topological shapes, the two types of trajectories require very different algorithms in the next stage of data reconstruction chain where pixels are clustered into individual particle trajectories. Prior to U-ResNet, mixing of two topological types was a major bottleneck. Distinction of two types prior to this reconstruction stage was enabled for the first time using U-ResNet, and is now utilized for the downstream reconstruction chain in the experiment [21].

U-ResNet in MicroBooNE is a supervised model that is trained using simulated image of particles. An assumption of this algorithm is that image features learned to perform the segmentation task are shared between data and simulation. A validation study is performed achieving percent-level statistical uncertainty [22]. As noted earlier, a study to confirm whether an implicit assumption of an algorithm holds or not is an important step for deploying a clustering algorithm. Beyond such a study, one may consider a strategy to actively mitigate discrepancies between the data and simulation domains. This challenge is called *domain adaptation* and is an active area of research in science including particle physics [38–40].

2.3. Scalable sparse segmentation for big data

The scalability of an algorithm to bigger data is an important aspect of ML methods in particle physics. Despite its successful application, because of both its encoder and decoder blocks, U-Net requires more computation time and memory resources compared to widely used CNNs for image classification. The MicroBooNE study reported that they cropped a whole detector image that consists of more than 13 million pixels into a smaller frame, 512×512 pixels, in order to run U-ResNet. This means a factor of 50 reduction in size, and requires an extra processing stage to stick back the image to the original size before running the downstream reconstruction tasks. This is clearly not ideal. Moreover, the majority of pixels shown in example image data are backgrounds and do not contain information about a particle trajectory (e.g. see Fig. 3), suggesting the majority of computation

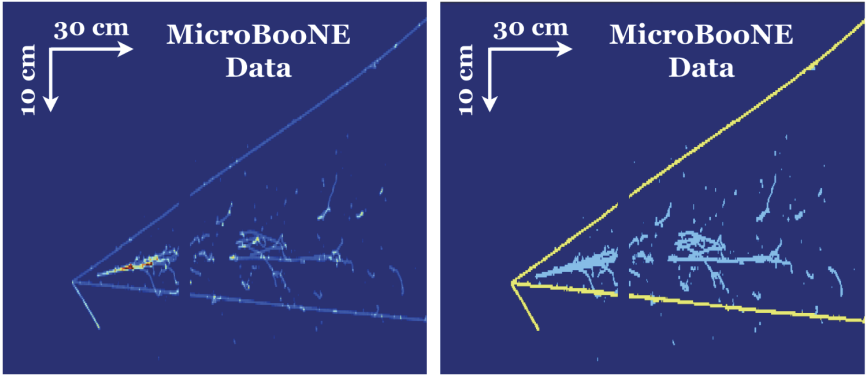


Fig. 3. Semantic segmentation applied in the MicroBooNE experiment to partition pixels into a track (yellow) and shower (cyan) categories. This figure is taken from [22].

performed may be wasted. U-Net with standard convolutional layers is not scalable to bigger image data such as DUNE far detector (DUNE-FD), which will have a much larger volume, corresponding to 40 kiloton of LAr, 450 times more than that of the MicroBooNE detector.

Recently, there have been two solutions proposed. The first is a class of sparse CNNs such as sparse submanifold convolutional networks [41, 42] and Minkowski CNNs [43]. These frameworks provide implementation of fast linear algebra for sparse tensors while keeping the nature of CNNs, which is to extract translation invariant features from an image using small kernels. A successful application of U-ResNet with a sparse CNN framework has been pioneered for 2D and 3D LArTPC image data [23] where orders of magnitude improvement is reported in both wall-time and memory required for computation. It is worth noting that those computational resources required for sparse CNNs scale almost linearly with the number of *active pixels* in an image [23] (i.e. those pixels that carry values and are not *null*), and does not depend on the spatial size of the image. This makes segmentation algorithms like U-Net scalable to data from larger imaging detectors and also to higher-dimensional data as it is already shown to work on 3D images.

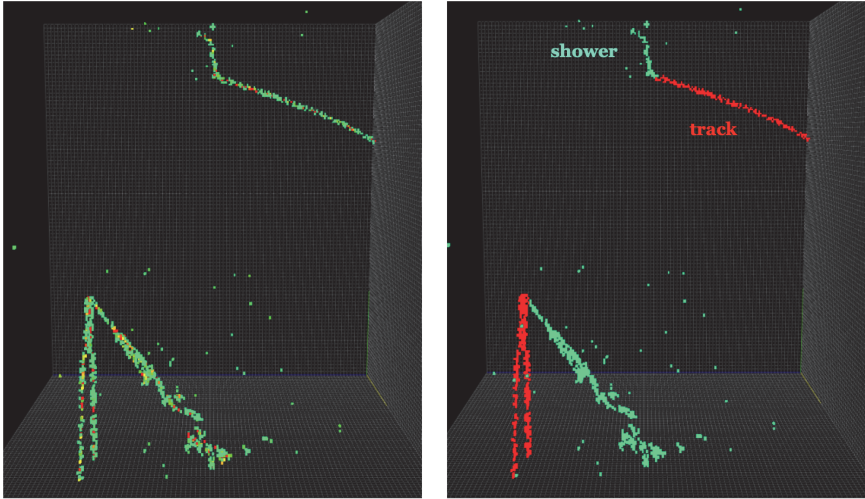


Fig. 4. The dynamic graph CNN applied to simulated particle trajectory in a LArTPC detector. Particle trajectories with energy depositions in a continuous color scale are shown on the left. The network output is shown on the right with two distinct colors, green and red for shower and track pixels respectively.

The second class of solutions consists of a graph neural network (GNN) [44]. In the case of semantic segmentation, an individual pixel could be interpreted as a graph *node*, and connected with neighboring nodes (i.e. neighboring pixels) through *edges*. This is analogous to a convolution operation in a CNN where a small kernel extracts translation invariant features. Figure 4 shows an output of a semantic segmentation task using the dynamic graph CNN (DGCNN) [45], implemented by the author [46]. The performance for semantic segmentation is comparable to that of sparse CNNs in terms of both computational resource and task accuracy.

The choice between sparse CNN and GNN depends on the application, and it is important to note key differences. First, GNNs can be considered as generalization of CNNs in a sense that they do not require data points to be arranged in a fixed-size grid format (i.e. a matrix), and also there is flexibility to define arbitrary edges between nodes. The latter is important as it defines neighboring nodes to be involved in a convolution operation. On the contrary,

CNNs act on matrix data and have much less flexibility in the convolutional kernels, which are typically a small rectangular matrix. It is also important to note that GNNs can effectively communicate information between nodes that are far apart in space by simply creating an edge between them. In contrast, in a CNN, an exchange of information between distant pixels is typically done through many downsampling operations, which also lacks fine spatial information as discussed earlier. While these facts appear advantageous for the use GNNs, the increased flexibility means more (potentially irrelevant) phase space to be explored during training. In other words, if applied for appropriate tasks, limitations for CNNs can be considered constraints motivated by domain knowledge or inductive bias. Furthermore, while similar constraints can be applied to GNNs by defining edges that only connect local neighbors, this step of graph preparation is not required for CNNs. Few systematic comparison studies have been performed, and none has been published to the knowledge of the author at the time of this writing.

2.4. *Application to Michel electron clustering*

For some particles, running DBSCAN on a semantic segmentation mask is sufficient to cluster pixels and reconstruct a trajectory. A Michel electron from a muon decay [47] in neutrino LArTPC detectors falls in this category as it is rare for two Michel electron trajectories to come in contact. As such, if a Michel electron is one of the semantic types, running DBSCAN on pixels labeled as Michel electrons by a semantic segmentation network would be sufficient to identify individual Michel electron trajectory. This is demonstrated by the authors of sparse U-ResNet paper [19] using 3D particle simulation images in LAr where the Michel electron identification purity and efficiency are reported as 97% and 93%, respectively. The pixel clustering purity and efficiency are both 96%. Figure 5 from their paper shows a good agreement in pixel counts between reconstructed and true Michel electron clusters, where the two classes of clusters are matched using a criteria of maximum overlap in shared pixels.



Fig. 5. Comparison of the pixel count between the true Michel electron clusters and the reconstructed ones. A reconstructed cluster is matched to a true cluster based on the maximum overlap of pixels between them. This figure is taken from [23].

2.5. Application to track clustering

Clustering of pixels into particle tracks, or trajectories, seems to be a simple task after those pixels are separated from shower particles that exhibit more complicated shapes. Since the trajectory of a track particle is simply a continuous line, DBSCAN seems to be a reasonable choice of algorithm to cluster its pixels. When two track particles originate from the same point (e.g. at an interaction vertex, or a decay point), it may need to be broken at the connection point. This is implemented in the work of the Point Proposal Network (PPN) [19, 20], which consists of a few sparse convolutional layers that work in conjunction with U-ResNet. The objective of PPN is the detection of endpoints of particle trajectories and positions regression of those points. Using the Region CNN (R-CNN) architecture, which has been known as one of the most successful architectures for detecting objects in an image, PPN can detect an arbitrary number of trajectory endpoints accurately. A proposed clustering algorithm is extremely simple. First, U-ResNet and PPN are run so that pixels are classified into track semantic types and endpoints are detected. Secondly, all pixels that are within seven pixels from all endpoints

are masked out, and DBSCAN is run to cluster individual particle tracks. Thirdly, masked pixels are put back, and assigned to the closest track cluster near the masking boundary. Figure 6 is taken from the PPN work [20] and visualizes these steps. This is a simple

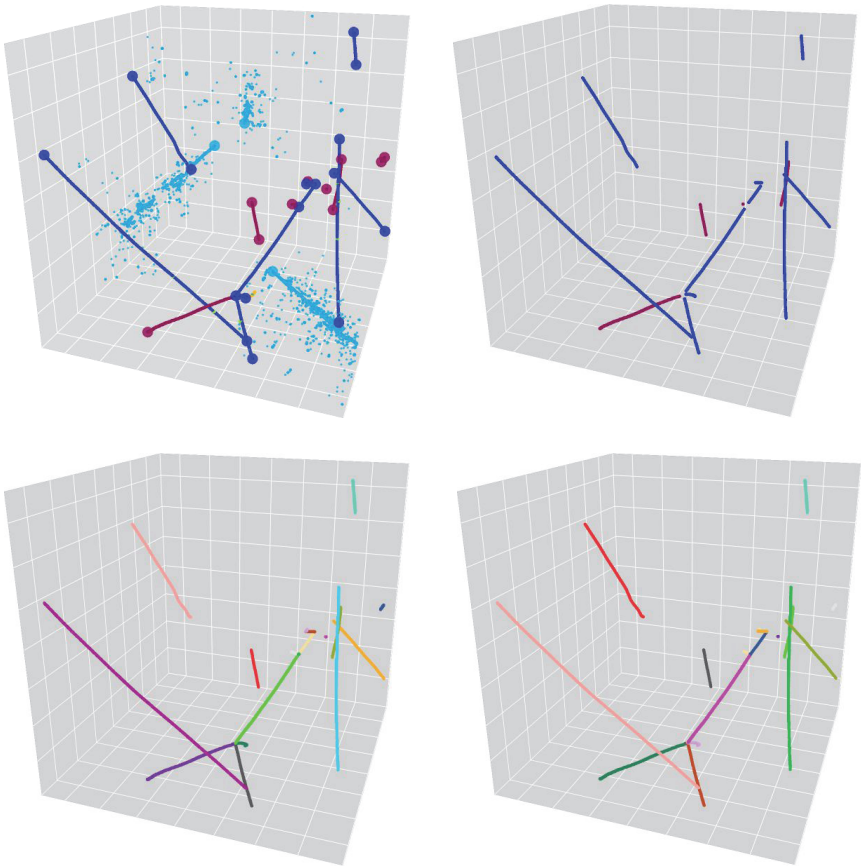


Fig. 6. Application of semantic segmentation for track clustering using DBSCAN and PPN. The segmentation output with the endpoints of particle trajectories, found by PPN, are shown on top-left. Then several pixels around those endpoints are masked out and DBSCAN is run for track pixels on the top right as an intermediate step. Distinct colors indicate distinct clusters in this and the bottom two images. Finally, the masked pixels are put back and assigned to the closest track cluster in the bottom right. The bottom left image shows the true underlying clusters. The coloring scheme is not meant to be identical between the bottom two images.

extension to an algorithm with a fixed partitions, like U-ResNet, to allow clustering of an arbitrary number of particle trajectories.

A shortcoming of DBSCAN is the assumption of a single-valued point density, which remains a concern in such a clustering algorithm. For LArTPC images, the thickness of a particle trajectory is known to depend on multiple factors including the diffusion of ionization electrons during drift and the angle of the trajectory with respect to the charge readout plane. Moreover, at the interaction vertex, where multiple particles may originate and is of the most interest for neutrino physics analysis, the density depends on the multiplicity and type of particles produced. When considering these factors, the point density near the particle endpoints is definitely not single-valued, and is more likely continuous. Aside from particle tracks, DBSCAN would not be a solution for particle showers whose tracks consist of widely varying sizes because of the 15–30 cm radiation length in LAr. ML algorithms that are capable of learning complex underlying image features are needed to address the challenge of pixel clustering in these LArTPC image data, and they are discussed in the following sections.

3. Convolutional Neural Networks for Pixel Clustering

Clustering of image pixels in order to identify an individual instance of an object in an image is a task called *instance segmentation* and has been an active area of research in the field of computer vision. Instance segmentation is a more advanced task than semantic segmentation with an additional step of partitioning pixels of the same class into separate instances. There are two well-known approaches to this task. The first is based on an object detector, such as R-CNN. This class of solution works in two steps: the first step detects individual objects' locations and sizes in the form of bounding boxes (see Fig. 7 for example), then the second step identifies pixels that belong to a target object within a bounding box. This class of solution is referred to as a *proposal-based approach*. The second class of solution is to use CNNs to learn a function that can transform

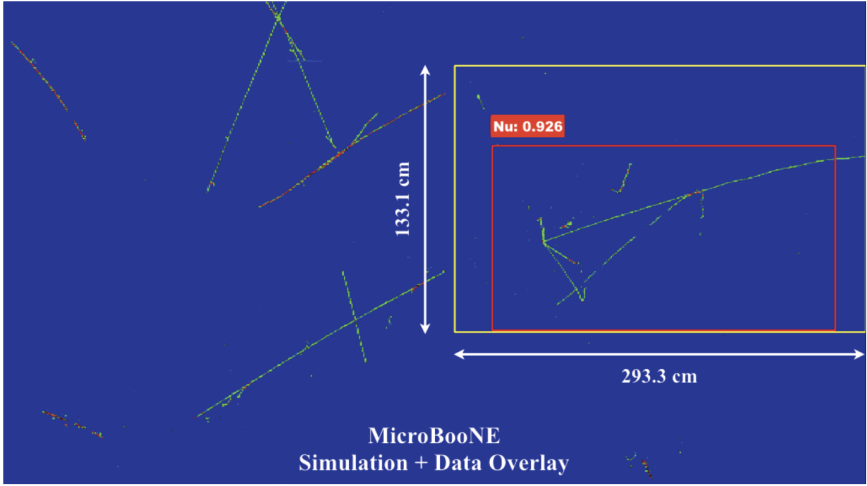


Fig. 7. Faster R-CNN object detection network applied in the MicroBooNE experiment. The location and size of a boundary box shown in red are produced by the algorithm while the true reference is shown in yellow. This figure is taken from [48, 49].

image pixels into a representation in an embedding space where the clustering task may be simplified. For instance, a loss function for a CNN may be conditioned with objectives similar to k -means by enforcing pixels that belong to the same particle trajectory gather close to each other in the embedding space, and ultimately to the same embedding coordinate. If successful, a simple algorithm such as DBSCAN may be employed in the embedding space to cluster pixels more easily.

3.1. *Region-proposal approach*

The region-proposal method consists of three components. The first is an encoder architecture for extracting image features. The second is a region proposal network (RPN) that detects an arbitrary number of object locations and proposes a bounding box, or region of interest (ROI), per instance. The third is a semantic segmentation network, such as FCN, which generates an instance *mask* by classifying all pixels within each ROI into the foreground (i.e. instance)

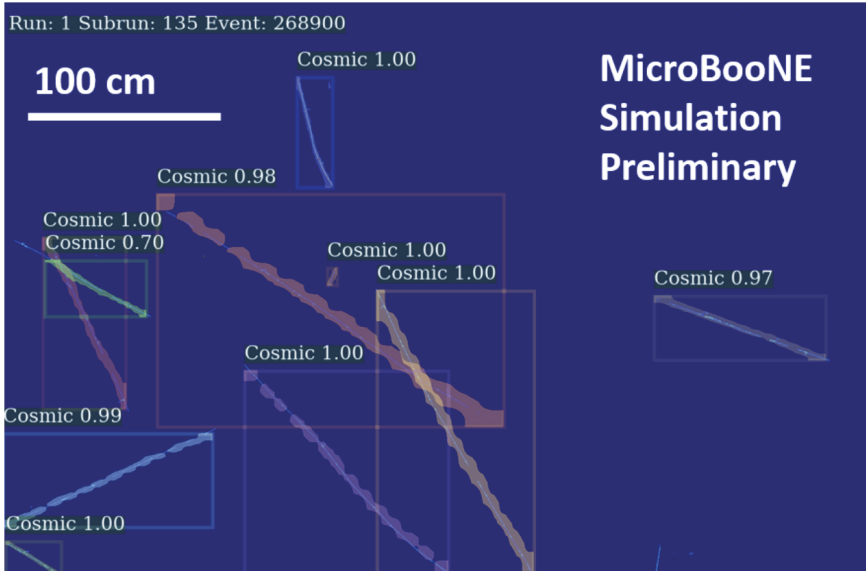


Fig. 8. Mask R-CNN applied in the MicroBooNE experiment to identify individual cosmic ray trajectories. A rectangular box indicates an identified particle. Partially transparent masks are generated by a masking network. This figure is taken from [50].

or the background. One of the most successful algorithms in this class is called mask R-CNN [3] which uses a faster R-CNN for an object detection with an FCN for instance masking. Mask R-CNN is widely used in computer vision and neutrino experiments including NOvA and MicroBooNE for identifying particle instances in an image. Figure 8 shows an implementation in the MicroBooNE experiment for identifying individual cosmic-ray trajectories [50], which are important backgrounds to be removed in their neutrino analysis.

Despite the success of Mask R-CNN in many applications, however, the approach using region-proposal mechanism is prone to a few challenges. First, as this technique restricts the segmentation challenge to the proposed region of interest, a misplaced bounding box has a direct consequence of a loss in clustering performance. If pixels that are part of the target object are not within its bounding box, they are simply lost. Second is an *object occlusion* issue: two

instances of the same type with bounding boxes of a similar position and size become inherently indistinguishable. Furthermore, as a domain-specific challenge, a rectangular bounding box is far from ideal to represent the region of a particle's trajectory and results in a large number of background pixels. It currently remains an active area of research to overcome these challenges and implement mask R-CNN and other region-proposal algorithms for clustering pixels in data reconstruction.

3.2. *Proposal-free approach*

Proposal-free approaches avoid restricting the segmentation challenge to proposed ROIs. One of the most successful solutions in this class is a pixel coordinate transformation, which maps pixel to an n -dimensional embedding space. Scalable proposal-free instance clustering in embedding (SPICE) has been introduced for analyzing 3D LArTPC image data [24]. It is based on sparse CNN with an architecture similar to U-ResNet, and is specifically for clustering densely connected pixels for which CNNs are well suited. SPICE transforms pixels from the 3D image space, $I \in \mathbb{Z}^3 \times \mathbb{R}$, represented by three integer coordinate values and one floating point pixel value, into the 3D embedding space \mathbb{R}^3 . Figure 9 shows the coordinates of pixels in both the image and the embedding space.

3.2.1. *Embedding loss*

The original work [2], from which SPICE is derived, employed the embedding loss, $\mathcal{L}_{\text{emb}} = \mathcal{L}_{\text{var}} + \mathcal{L}_{\text{int}} + \mathcal{L}_{\text{reg}}$, in order to condition the transformation process. \mathcal{L}_{var} represents the variance of pixel coordinates with respect to the centroid of the cluster to which they belong. \mathcal{L}_{int} concerns the distance between the centroids of clusters, and rewards the algorithm for keeping a certain intercluster distance. Finally, \mathcal{L}_{reg} is a linear sum of the distances to the centroid from the origin of the embedding space, which acts as a L_1 regularization loss to place clusters near the origin. The effects of \mathcal{L}_{var} and \mathcal{L}_{int} are visualized in Fig. 10 from the original work [2]. These essentially work as

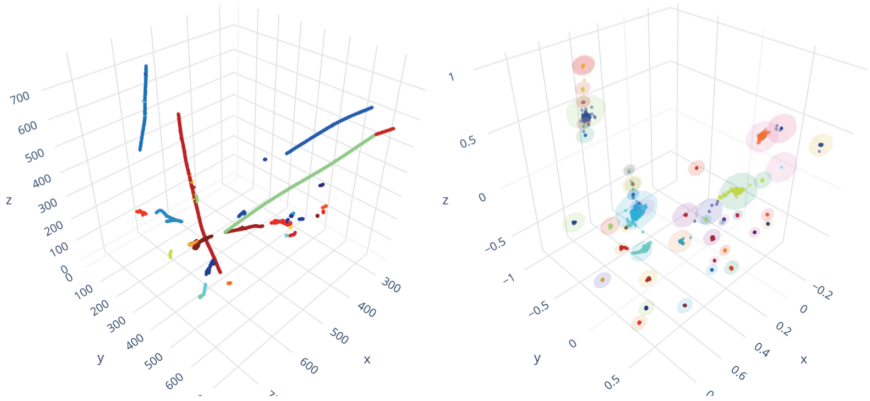


Fig. 9. Transformation of pixels from the 3D image space (left) into the 3D embedding space (right) by SPICE. Pixel colors are discrete, and those pixels that belong to the same particle trajectory (cluster) share the same color.

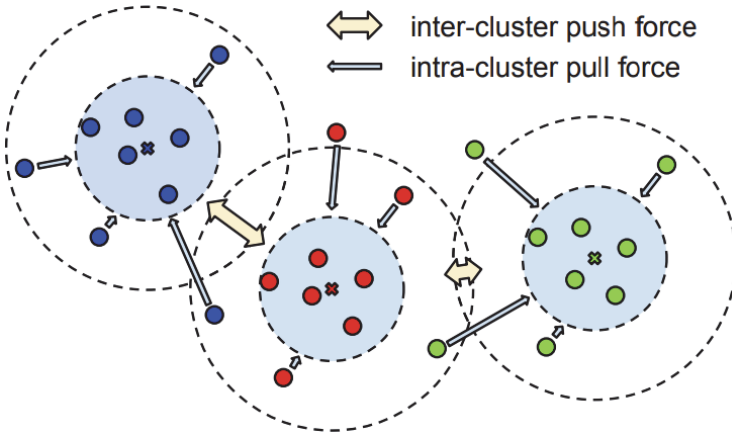


Fig. 10. \mathcal{L}_{var} and \mathcal{L}_{int} act as a force to pull pixels to the cluster centroid and to repel centroids of different clusters, respectively. This figure is taken from [2].

an attractive force to gather pixels to the centroid of their clusters and a repulsive force between the centroids of different clusters.

While these are intuitive conditions for defining the transformation, there still needs to be an actual pixel clustering step in the embedding space. In the original work, mean-shift [51] was employed, which is not a learnable algorithm. The shortcoming of this approach

is that the actual clustering step in the post-processing is not a part of the algorithm optimization, and hence the embedding space may become suboptimal.

3.2.2. *Joint optimization of embedding and clustering*

More recent work [4] incorporates the pixel clustering step and its loss function so that the whole pipeline of pixel clustering can be optimized end-to-end. In this work, in addition to the coordinates in the embedding space, two additional parameters are estimated by the algorithm for every pixel: a cluster *margin* and the *seediness* of a pixel. The seediness is a measure of how likely it is that a pixel may represent the centroid of the cluster to which it belongs in the embedding space. The margin loss function is the spread of pixels with respect to the centroid of the cluster to which a pixel may belong. The margin loss is conditioned to minimize the spread of margin values across pixels that belong to the same cluster. However, a margin value is allowed to vary among clusters to accommodate the fact that there are variations in the size and confidence of the cluster (i.e. “difficult” clusters may have a larger margin). Finally, the embedding loss function is conditioned such that the pixels follow a normalized Gaussian distribution around the centroid of a cluster. In the inference, pixels are ordered from high to low seediness scores. Given a candidate seed pixel, a score that indicates whether or not another pixel belongs to a candidate cluster represented by this seed pixel can be calculated using the seed pixel’s embedding coordinate and margin as the Gaussian mean and standard deviation, respectively. This allows the algorithm to assign pixels to the highest score cluster.

SPICE combines this preceding research [2, 4] with the U-ResNet architecture as a backbone as shown in Fig. 11. The decoder is split into two branches where one branch is responsible for learning a transformation function into the embedding space and the margin value, and the other branch estimates the seediness score. An example output from their study [24] is shown in Fig. 12. The adjusted Rand index [52], a standard metric for pixel clustering in image analysis, is above 0.98 for this image, which indicates nearly perfect

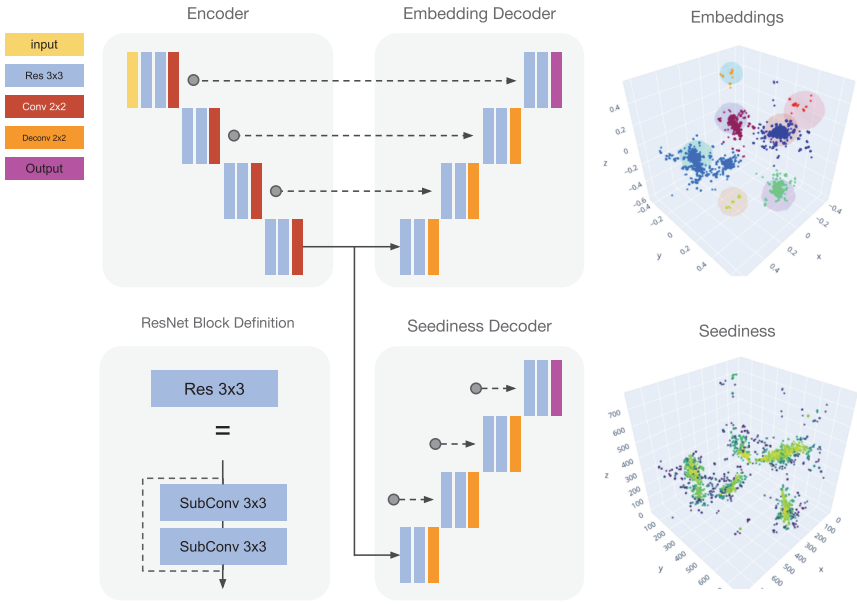


Fig. 11. The SPICE network architecture, shown on the left, follows the U-ResNet design, using residual connections and sparse convolutional layers, with one shared encoder and multiple decoder branches. The coordinates and seediness scores of pixels in the embedding space are shown on the right. This figure is taken from [24].

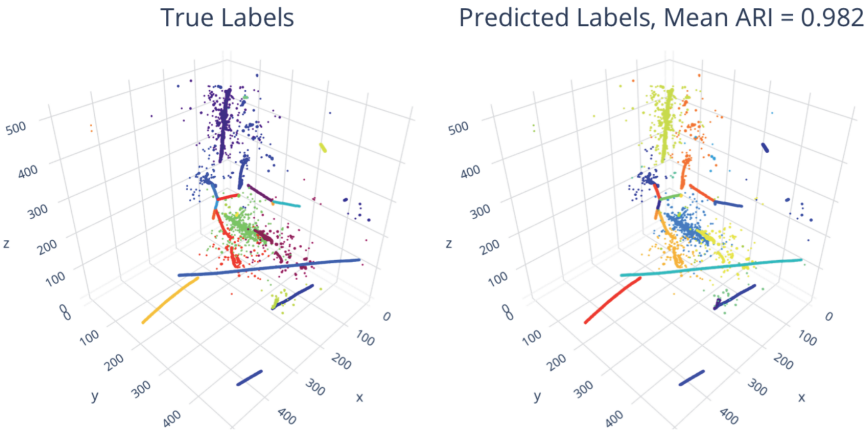


Fig. 12. Comparison of true underlying clusters (left) and the clusters reconstructed (right) by SPICE. Discrete colors are assigned to the pixels that belong to the same cluster. This figure is taken from [24].

clustering. SPICE is the first purely 3D pixel clustering algorithm proposed for LArTPC experiments where the interaction vertex is unknown and can be anywhere in the detector. It is expected to be used for analyzing 3D image data from the DUNE near detector (DUNE-ND), a pixel-based 3D imaging LArTPC.

Despite the successful introduction of SPICE, which is meant to overcome the challenge of object occlusions faced by proposal-based approaches, it should be noted that SPICE is subject to its own limitations because it uses CNNs. For example, consider a muon with a very long trajectory. For an algorithm like SPICE to transfer all pixels from the image space to the centroid of a cluster in the embedding space, the pixels *somehow* need to communicate a common target location. Yet, the distance in the image space across which pixels can communicate, called the size of the *receptive field*, depends on the CNN architecture as it requires convolution or downsampling operations. The limitation ultimately comes from the fact that CNN kernels only connect, or combine features from, neighboring pixels. This could be overcome by using GNNs, in which edges can be defined between distant nodes to enable long-distance information propagation without extra operations, while keeping the same objectives and loss function definitions employed by SPICE. We shall focus on *particle clustering* in the next section. This advantage becomes more apparent and the choice of GNN as an underlying architecture is more natural.

4. Clustering Particles Using Graph Neural Networks

Clustering in particle physics data reconstruction is naturally a hierarchical task. For example, pixels may be clustered into particle trajectories, and then some particles may be clustered as originating from a single interaction. An intermediate particle representation may also be useful, such as an electromagnetic shower that consists of many trajectories of individual electromagnetic particles, or a neutral pion represented by two decay photons. While CNNs provide a natural representation for matrix-formatted image data, a generalization is needed for a higher-level data representations such as clusters.

A graph representation, in which constituents and their connections are represented as nodes and edges, respectively, provides generalized input and output data capable of specifying clustering tasks at different levels.

With GNNs, a clustering task can be solved in the form of binary edge classifications. Simply put, nodes connected by *valid* edges belong to the same cluster, while disconnected nodes, or those connected by *invalid* edges, do not. While this sounds simple, implementations vary wildly due to the flexibility in designing the GNNs. Important considerations include the initial graph construction, operations to extract node and edge features from input data, static vs. dynamic graph (i.e. a dynamic graph may generate or erase new or existing nodes or edges), operations for updating node and edge features (known as *message passing*: a mechanism to propagate information throughout the graph), and lastly a post-processing or interpretation of the graph nodes and edges in the final state. This section focuses on a comprehensive survey of GNN architectures for LArTPC particle and interaction clustering applications [25]. Other notable examples include clustering of hits for tracking and calorimeters in the ATLAS and CMS experiments respectively [53], and they are covered by other authors in Chapter 12.

4.1. *Clustering electromagnetic shower fragments*

An electromagnetic shower may contain many small fragments of electron and positron trajectories that need to be clustered together. The radiation length is large (15–30 cm) as compared to the millimeter per pixel image resolution, which causes a gap between fragments of varying size, from a few to hundreds of pixels. Following the success of highly accurate semantic segmentation techniques [19], a GNN solution [25] has been proposed and has demonstrated promising performance. First, DBSCAN is run only on the shower pixels identified by U-ResNet, a semantic segmentation model. This produces many small fragments of electromagnetic showers as shown in Fig. 13, taken from the original paper. A graph is constructed by taking each fragment as a node and a connection between fragments as an edge.

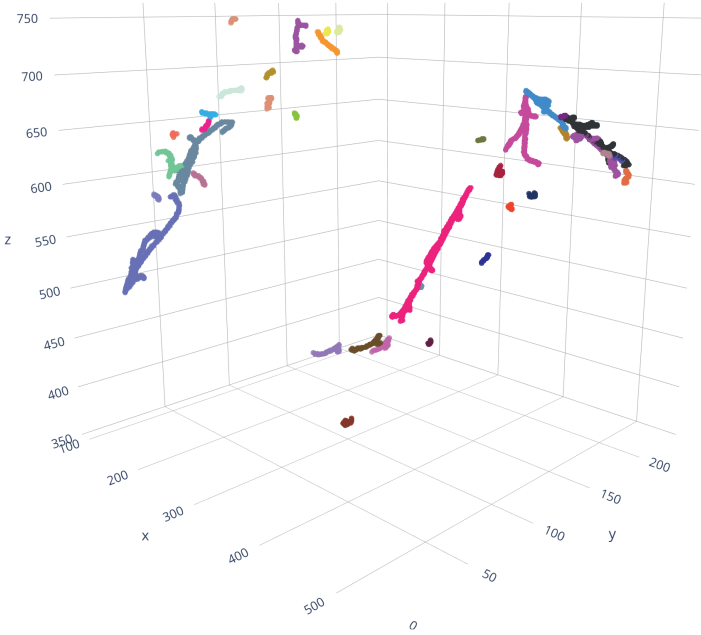


Fig. 13. Shower fragments produced by running DBSCAN on shower pixels. The color scale is discrete and pixels that belong to the same fragment cluster share the same color. This figure is taken from [25].

In addition to the edge classification, a GNN in this study is trained jointly for a node classification in order to identify a *primary* node, or the root fragment of the shower. This is important as the primary fragment is the most informative in terms of the starting position and the initial direction of the shower. The authors of this work explored a variety of options at all stages of problem solving including:

- input graph construction methods including a complete graph, edges in Delaunay triangulation, minimum spanning tree (MST), and five nearest neighbors;
- encoding methods for the initial node and edge features including geometrical features, addition of shower starting position from PPN, a sparse CNN encoder as a learnable feature extractor from the pixel level;

- message passing mechanisms including a full graph network [54], neural message passing [55], DGCNN [56], and a graph attention network [57];
- the number of iterations of a message passing operation;
- two target graph definitions including a cluster graph, a disjoint union of a complete graph, and a “forest”, a collection of trees where each tree represents a particle flow within each shower.

Skipping the discussion details, which can be found in their paper, the recommended choices for this clustering task are a complete graph as an input, geometrical features with the shower starting position from PPN to define the initial node and edge features, a full graph neural network for message passing, three rounds of message passing, and a cluster graph as an optimization target. Figure 14 shows an example inference output under this configuration. Overall, the reported clustering purity and efficiency are both 99.5%, and the GNN achieves 97.7% for the adjusted Rand index [52] (ARI) clustering performance metric. The node classification accuracy to identify the primary fragment of a shower is reported as 99.77%.

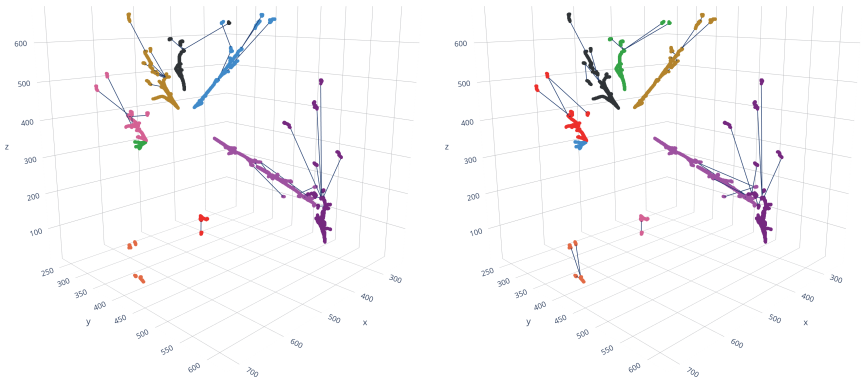


Fig. 14. Left: the true underlying clusters (color) of electromagnetic shower fragments with true edges defined based on the flow of particles. Right: the inferred clusters and edges using a GNN. This figure is taken from [25].

4.2. Loss function

Two observations in the study of electromagnetic shower clustering [25] are worth noting. The first is a comparison of two target cluster types: a cluster graph and a forest. A derivation of the latter is a more challenging task as the edges in a correct tree are a subset of the edges in a cluster graph, hence requiring more discrimination power. In turn, a forest prediction provides not only a cluster, but also a particle flow within a shower. As a pleasant surprise, the final clustering performance between two target graphs are nearly identical as shown in Fig. 15. This seems promising to extend the same approach for a generic particle flow reconstruction using a GNN with directed edges.

The second point is an obvious, yet an important remark that is often missed. Quoting exactly from the paper, “*The network predicts an edge score matrix, \mathbf{S}^e , which tries to replicate the predefined ground-truth adjacency matrix, \mathbf{A} . In a graph partition problem, \mathbf{A} should be designed such that, if $a_{ij} = 1$, then nodes i and j belong to the same group. The converse statement does not have to hold,*

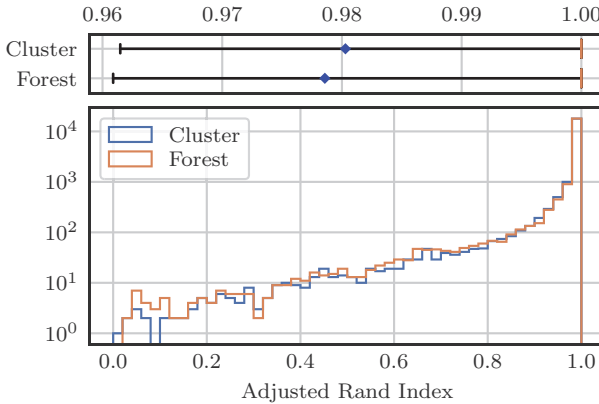


Fig. 15. Comparison of different target graphs, a cluster graph vs. a forest, using the ARI metric for clustering of electromagnetic showers. This figure is taken from [25].

as nodes i and j may not be connected directly as long as they are linked through an indirect path.” In other words, running an inference simply by selecting graph edges based on individual scores may not result in the most optimal graph, or equivalently, that may result in a larger loss function value. In the paper, the authors give a further clarification using two cluster graphs connected by two edges with score values 0.1 and 0.6. If one naively interprets the edge with a score of 0.6 as a valid edge, this results in merging the two clusters, which is equivalent to selecting the other edge despite the lower score of 0.1. As a result, the overall loss may become larger than two disjoint cluster graphs. Without this consideration, the approach using a cluster target is much more prone to incorrectly merging two clusters because of the presence of any edge between them with a score above 0.5.

4.3. *Clustering interactions*

GNNs can provide a generalized clustering framework as noted above, and this is demonstrated by applying the same architecture at the next stage of the reconstruction chain: the clustering of particles that share the same origin interaction. At this stage, a graph node is a particle of any type, including both track and shower particles. A graph pooling operation may be used to aggregate features from clustered fragments for shower particles. This is, however, not done in the paper, and instead the node and edge features are derived using similar methods employed at the shower clustering stage, namely geometrical features computed from analytical functions, PPN output, and a particle’s semantic type from U-ResNet. An example output image is shown in Fig. 16.

Due to the high-intensity neutrino beam for the DUNE program, DUNE-ND is expected to observe a pile-up of more than 20 neutrino interactions per image for the first time in the history of experimental neutrino physics. It is one of the hardest data reconstruction challenges yet to be addressed. The GNN-based particle clustering technique shows a strong promise to address this challenge. In [25], the mean values of a purity, efficiency, and ARI of particle clustering

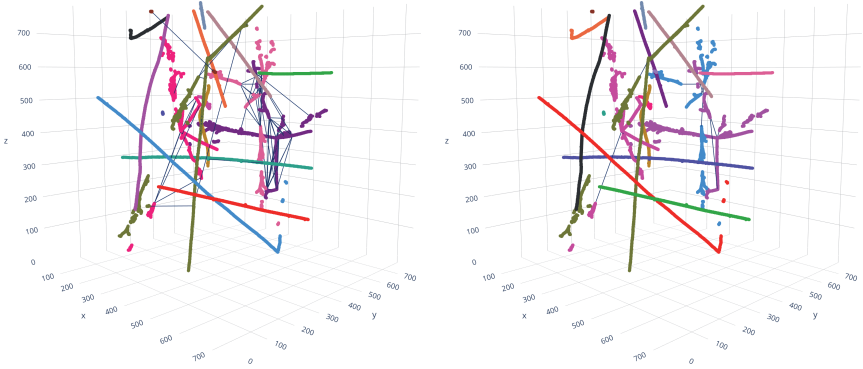


Fig. 16. Left: the true underlying clusters (color) of particles with true edges defined based on the particle flow. Right: the inferred clusters and edges using GNN. The ARI for this randomly selected event is 100% (perfect). This figure is taken from [25].

are all above 99% for images in which particle density per unit volume is comparable to DUNE-ND.

5. Summary

Clustering is a critical part of data reconstruction tasks in particle physics, and its R&D is an active area at the interface of ML and many domain sciences. While traditional, unsupervised clustering techniques have utilized for a long time, learnable methods based on CNNs and GNNs have been developed with promising results recently. We have covered semantic segmentation as a method to partition image pixels into a set of predefined categories. Pioneering algorithms, including U-Net and FCN, have been utilized in identifying different types of particle trajectories. However, many techniques in computer vision have been developed for natural images, and applying them for data from particle physics detectors would need to address domain specific challenges. For instance, sparse CNNs and GNNs have been used to adopt semantic segmentation methods for LArTPC images that are globally sparse, yet contain densely sampled particle tracks.

Instance segmentation takes another task of identifying individual instances within each semantic type. We discussed two approaches: proposal-based and proposal-free methods. Mask R-CNN is an example of the former and works in two steps including identification of a bounding box per instance and generation of a pixel-level mask inside each box. SPICE is a proposal-free based method and performs a transformation of pixels from input image space into the embedding space where clustering can be more easily performed in the post processing. GNNs can be an effective solution for clustering many objects that are separated by arbitrary distances. In [25], the same GNN is used to reconstruct individual electromagnetic showers and to cluster particles in the same neutrino interaction.

Advancements from the computer vision community and interdisciplinary collaboration with domain sciences are expected to continue pushing the boundaries. GNNs are a versatile solution to many, if not all, clustering tasks in data reconstruction. The high degree of flexibility in designing the graph structure and communication methods between nodes and edges may be exploited to introduce more physics domain knowledge. Extraction of features from multi-modal particle detector data can be a unique challenge. Implementing deep knowledge of physics models into ML algorithms may be a complicated task that requires new developments. However, these challenges make the field of experimental particle physics an exciting domain to pursue the development of ML techniques for clustering.

Acknowledgement

This work was supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, and Early Career Research Program under Contract DE-AC02-76SF00515.

References

- [1] S. Ren, K. He, R. Girshick and J. Sun, Faster R-CNN: Towards real-time object detection with region proposal networks, in *Advances in Neural Information Processing Systems (NIPS)* (2015).

- [2] B. De Brabandere, D. Neven and L. Van Gool, Semantic instance segmentation for autonomous driving, in *The IEEE Conf. Computer Vision and Pattern Recognition (CVPR) Workshops* (July, 2017).
- [3] K. He, G. Gkioxari, P. Dollár and R. Girshick, Mask r-cnn, in *2017 IEEE International Conf. Computer Vision (ICCV)* (2017), pp. 2980–2988.
- [4] D. Neven, B. D. Brabandere, M. Proesmans and L. Van Gool, Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth, in *2019 IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)* (2019); pp. 8829–8837.
- [5] scikit developers, Clustering (2007); <https://scikit-learn.org/stable/module/s/clustering.html>.
- [6] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise, in *Proc. Second Int. Conf. Knowledge Discovery and Data Mining, KDD'96* (AAAI Press, 1996), pp. 226–231.
- [7] E. D. Church, LArSoft: A software package for liquid argon time projection drift chambers (2013); arXiv:1311.6774.
- [8] S. Amrouche *et al.*, The tracking machine learning challenge: Accuracy phase, in S. Escalera and R. Herbrich (eds.), *The NeurIPS '18 Competition* (2020), pp. 231–264; doi:10.1007/978-3-030-29135-8_9.
- [9] M. Ankerst, M. Breunig, H.-P. Kriegel and J. Sander, Optics: Ordering points to identify the clustering structure, *ACM Sigmod Record* **28**(2) (1999) 49–60.
- [10] R. J. G. B. Campello, D. Moulavi and J. Sander, Density-based clustering based on hierarchical density estimates, in *Advances in Knowledge Discovery and Data Mining*, eds. J. Pei, V. S. Tseng, L. Cao, H. Motoda and G. Xu, (Springer, Berlin, 2013), pp. 160–172.
- [11] J. MacQueen, Some methods for classification and analysis of multivariate observations, in *Proc. Fifth Berkeley Symp. Mathematical Statistics and Probability, Volume 1: Statistics* (University of California Press, Berkeley, CA, 1967), pp. 281–297.
- [12] A. P. Dempster, N. M. Laird and D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, *J. Royal Statist. Soc. Ser. B (Methodological)* **39**(1) (1977) 1–38.
- [13] D. Arthur and S. Vassilvitskii, K-means++: The advantages of careful seeding, in *Proc. 18th Annual ACM-SIAM Symp. Discrete Algorithms* (2007).
- [14] ATLAS Collaboration, The ATLAS experiment at the CERN Large Hadron Collider, *J. Instrum.* **3**(08) (2008) S08003; doi:10.1088/1748-0221/3/08/S08003.
- [15] A. Salzburger, Optimisation of the ATLAS track reconstruction software for Run-2, *J. Phys. Conf. Ser.* **664**(7) (2015) 072042; doi:10.1088/1742-6596/664/7/072042.
- [16] ATLAS Collaboration, Performance of the ATLAS track reconstruction algorithms in dense environments in LHC Run 2, *Eur. Phys. J. C* **66**(673) (2017); doi:10.1140/epjc/s10052-017-5225-7.

- [17] J. Alves *et al.*, The LHCb Detector at the LHC, *J. Instrum.* **3** (2008) S08005; doi:10.1088/1748-0221/3/08/S08005.
- [18] M. Stahl, Machine learning and parallelism in the reconstruction of LHCb and its upgrade, *J. Phys. Conf. Ser.* **898** (2017) 042042; doi:10.1088/1742-6596/898/4/042042.
- [19] L. Domine and K. Terao, Applying deep neural network techniques for LArTPC data reconstruction (2018); doi:10.5281/zenodo.1300713.
- [20] L. Dominé, P. C. de Soux, F. Drielsma, D. H. Koh, R. Itay, Q. Lin, K. Terao, K. V. Tsang and T. L. Usher, Point proposal network for reconstructing 3D particle endpoints with sub-pixel precision in liquid argon time projection chambers, *Phys. Rev. D* **104** (2021) 032004.
- [21] MicroBooNE Collaboration, Vertex-finding and reconstruction of contained two-track neutrino events in the microboone detector, *J. Instrum.* **16** (2021) P02017.
- [22] MicroBooNE Collaboration, Deep neural network for pixel-level electromagnetic particle identification in the microboone liquid argon time projection chamber, *Phys. Rev. D* **99** (2019) 092001; doi:10.1103/PhysRevD.99.092001.
- [23] L. Dominé and K. Terao, Scalable deep convolutional neural networks for sparse, locally dense liquid argon time projection chamber data, *Phys. Rev. D* **102** (2020) 012005; doi:10.1103/PhysRevD.102.012005.
- [24] D. H. Koh, P. C. de Soux, L. Dominé, F. Drielsma, R. Itay, Q. Lin, K. Terao, K. V. Tsang and T. Usher, Scalable, proposal-free instance segmentation network for 3D pixel clustering and particle trajectory reconstruction in liquid argon time projection chambers (2020); arXiv:2007.03083.
- [25] F. Drielsma, Q. Lin, P. C. de Soux, L. Dominé, R. Itay, D. H. Koh, B. J. Nelson, K. Terao, K. V. Tsang and T. L. Usher, Clustering of electromagnetic showers and particle interactions with graph neural networks in liquid argon time projection chambers data (2020); arXiv:2007.01335.
- [26] MicroBooNE Collaboration, Design and construction of the MicroBooNE detector, *J. Instrum.* **12**(02) (2017) P02017; doi:10.1088/1748-0221/12/02/p02017.
- [27] S. Amerio *et al.*, Design, construction and tests of the icarus t600 detector, *Nucl. Instrum. Methods Phys. Res. A* **527**(3) (2004) 329–410; doi:10.1016/j.nima.2004.02.044.
- [28] DUNE Collaboration, Long-Baseline Neutrino Facility (LBNF) and Deep Underground Neutrino Experiment (DUNE) (2016), arXiv:1601.05471.
- [29] C. Adams, K. Terao and T. Wongjirad, Pilarnet: Public dataset for particle imaging liquid argon detectors in high energy physics (2020); arXiv:2006.01993.
- [30] J. Long, E. Shelhamer and T. Darrell, Fully convolutional networks for semantic segmentation (2014); arXiv:1411.4038.
- [31] O. Ronneberger, P. Fischer and T. Brox, U-net: Convolutional networks for biomedical image segmentation (2015); arXiv:1505.04597.
- [32] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, in *Shape*,

- Contour and Grouping in Computer Vision*, Vol. 36 (Biological Cybernetics, 1980), pp. 193–202; doi:10.1007/BF00344251.
- [33] Y. LeCun, P. Haffner, L. Bottou and Y. Bengio, Object recognition with gradient-based learning, in *Shape, Contour and Grouping in Computer Vision* (Springer, Berlin, 1999), pp. 319–345; doi:10.1007/3-540-46805-6_19.
 - [34] A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems 25*, eds. F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger (Curran Associates, Inc., 2012), pp. 1097–1105.
 - [35] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition (2014); arXiv:1409.1556.
 - [36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9; doi:10.1109/CVPR.2015.7298594.
 - [37] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778; doi:10.1109/CVPR.2016.90.
 - [38] M. Wang and W. Deng, Deep visual domain adaptation: A survey, *Neuro-computing* **312** (2018) 135–153.
 - [39] G. Louppe, M. Kagan and K. Cranmer, Learning to pivot with adversarial networks, in *Advances in Neural Information Processing Systems*, Vol. 30, pp. 981–990, 2017; arXiv:1611.01046.
 - [40] MINERvA Collaboration, Reducing model bias in a deep learning classifier using domain adversarial neural networks in the MINERvA experiment, *J. Instrum.* **13**(11) (2018) P11020; doi:10.1088/1748-0221/13/11/p11020.
 - [41] B. Graham and L. van der Maaten, Submanifold sparse convolutional networks (2017); arXiv:1706.01307.
 - [42] B. Graham, M. Engelcke and L. van der Maaten, 3D semantic segmentation with submanifold sparse convolutional networks, in *Proc. IEEE Computer Vision and Pattern Recognition CVPR* (Salt Lake City, UT, USA, 2018), pp. 18–22; doi:10.1109/CVPR.2018.00961.
 - [43] C. Choy, J. Gwak and S. Savarese, 4D spatio-temporal convnets: Minkowski convolutional neural networks, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2019), pp. 3075–3084.
 - [44] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li and M. Sun, Graph neural networks: A review of methods and applications, *AI Open* **1** (2020) 57–81.
 - [45] F. Manessi, A. Rozza and M. Manzo, Dynamic graph convolutional networks, *Pattern Recognition* **97** (2020) 107000; doi:10.1016/j.patcog.2019.107000.
 - [46] K. T. Laura Domine, Semantic segmentation using dynamic graph CNN (2018), URL <https://github.com/DeepLearnPhysics/dynamic-gcnn>.
 - [47] L. Michel, Interaction between four half-spin particles and the decay of the meson, *Proc. Phys. Soc. Sect. A* **63**(5) (1950) 514–531; doi:10.1088/0370-1298/63/5/311.

- [48] MicroBooNE Collaboration, Convolutional neural networks applied to neutrino events in a liquid argon time projection chamber, *J. Instrum.* **12**(03) (2017) P03011; doi:10.1088/1748-0221/12/03/p03011.
- [49] A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, H. Alexander, A. Aurisano, K. Terao and T. Wongjirad, Machine learning at the energy and intensity frontiers of particle physics, *Nature* **560** (2018) 41–48; doi:10.1038/s41586-018-0361-2.
- [50] MicroBooNE Collaboration, Cosmic muon clustering for the MicroBooNE liquid argon time projection chambers using sMask-RCNN (2020). URL <https://microboone.fnal.gov/wp-content/uploads/MICROBOONE-NOTE-1081-PUB.pdf>.
- [51] D. Comaniciu and P. Meer, Mean shift: A robust approach toward feature space analysis, *IEEE Trans. Pattern Anal. Machine Intell.* **24**(5) (2002) 603–619.
- [52] L. Hubert and P. Arabie, Comparing partitions, *J. Classification* **2**(1) (1985) 193–218; doi:10.1007/BF01908075.
- [53] X. Ju, S. Farrell, P. Calafiura, D. Murnane, Prabhat, L. Gray, T. Klijnsma, K. Pedro, G. Cerati, J. Kowalkowski, G. Perdue, P. Spentzouris, N. Tran, J.-R. Vlimant, A. Zlokapa, J. Pata, M. Spiropulu, S. An, A. Aurisano, J. Hewes, A. Tsaris, K. Terao and T. Usher, Graph neural networks for particle reconstruction in high energy physics detectors (2020); arXiv:2003.11603.
- [54] P. W. Battaglia *et al.*, Relational inductive biases, deep learning, and graph networks (2018); arXiv:1806.01261.
- [55] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals and G. E. Dahl, Neural message passing for quantum chemistry, in *Proc. 34th Int. Conf. Machine Learning*, Vol. 70 (2017), p. 1263.
- [56] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein and J. M. Solomon, Dynamic graph CNN for learning on point clouds (2018); arXiv:1801.07829.
- [57] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, Graph attention networks, in *6th Int. Conf. Learning Representations* (2018).

Chapter 12

Graph Neural Networks for Particle Tracking and Reconstruction

Javier Duarte* and Jean-Roch Vlimant†

* *University of California San Diego, La Jolla, CA 92093, USA*
jduarte@ucsd.edu

† *California Institute of Technology, Pasadena, CA 91125, USA*
jvlimant@caltech.edu

Machine learning methods have a long history of applications in high-energy physics (HEP). Recently, there is a growing interest in exploiting these methods to reconstruct particle signatures from raw detector data. In order to benefit from modern deep learning algorithms that were initially designed for computer vision or natural language processing tasks, it is common practice to transform HEP data into images or sequences. Conversely, graph neural networks (GNNs), which operate on graph data composed of elements with a set of features and their pairwise connections, provide an alternative way of incorporating weight sharing, local connectivity, and specialized domain knowledge. Particle physics data, such as the hits in a tracking detector, can generally be represented as graphs, making the use of GNNs natural. In this chapter, we recapitulate the mathematical formalism of GNNs and highlight aspects to consider when designing these networks for HEP data, including graph construction, model architectures, learning objectives, and graph pooling. We also review promising applications of GNNs for particle tracking and reconstruction in HEP and summarize the outlook for their deployment in current and future experiments.

1. Introduction

Since the 1980s, machine learning (ML) techniques, including boosted decision trees, support vector machines, cellular automata, and multilayer perceptrons, have helped shape experimental particle

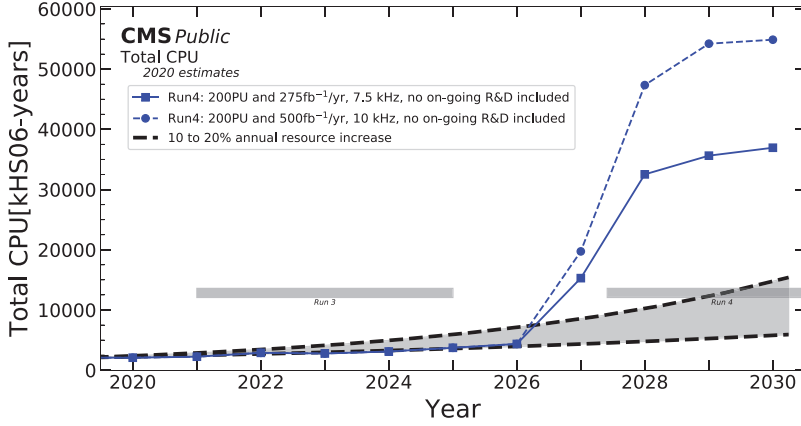
physics [1, 2]. As deep neural networks have achieved human-level performance for various tasks such as object recognition in images, they have been adopted in the physical sciences [3] including particle physics. Unlike traditional approaches, deep learning techniques operate on lower-level information to extract higher-level patterns directly from the data. Applications of ML in high-energy physics (HEP) have skyrocketed in recent years [2, 4–7]. However, until recently it was necessary to completely transform HEP data into images or sequences in order to use modern deep learning algorithms that were initially designed for computer vision or natural language processing tasks.

Geometric deep learning (GDL) [8–14] is a growing subfield of artificial intelligence (AI) that studies techniques generalizing structured deep neural network models to non-Euclidean domains such as sets, graphs, and manifolds. This includes the study of graph neural networks (GNNs) that operate on graph data composed of elements with a set of features, and their pairwise connections. Extensive reviews of GNNs are available in [11, 14–19] that provide in-depth technical details of current models.

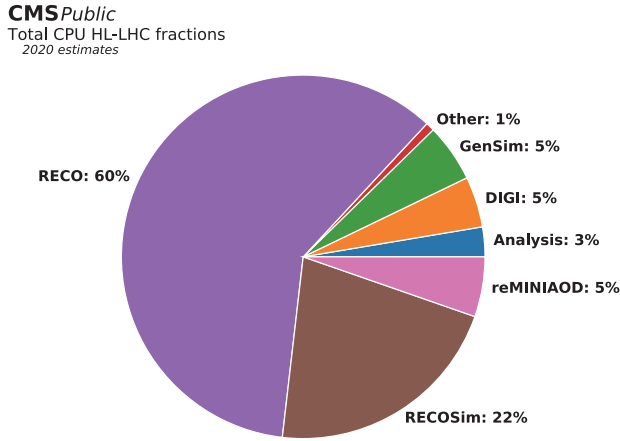
As the data from particle physics experiments are generally sparse samplings of physics processes in time and space, they are not easily represented as regular-grid images or as ordered sequences. Moreover, to reconstruct the input measurements into target particles, there is not always a clean, one-to-one mapping between the set of measurements and the set of particles because one particle can leave multiple traces in different subdetectors (many-to-one) and multiple particles can contribute to the same signal readout (one-to-many). GDL algorithms, including GNNs, are well-suited for this type of data and event reconstruction tasks. Unlike fully-connected (FC) models, convolutional neural networks (CNNs), and recurrent neural networks (RNNs), GNNs fully exploit the relational structure of the data. Recent work has applied set- and graph-based architectures in the domain of particle physics to charged particle tracking [20–25], jet classification [26–33] and building [34, 35], event classification [36–38], clustering [21, 39], vertexing [40, 41], particle finding [42, 43], and pileup mitigation [44, 45]. Many of these applications are reviewed in [46].

Analyses in particle physics are usually performed on high-level features, abstracted from the low-level detector signals. The distillation of the raw detector data into a physics-centric representation is called *reconstruction*, and is traditionally done in multiple stages — often at different levels of abstraction that physicists can naturally comprehend. A classic reconstruction algorithm, by design, may be limited in how much detail and information is used from the data, often to simplify its commissioning and validation. Conversely, an algorithm based on ML can learn directly from the full complexity of the data and thus may potentially perform better. This effect is well illustrated in the sector of *jet tagging* (see Chapter 13), where ML has brought significant improvements [6]. GNNs, because of the relational inductive bias they carry, have a great deal of expressive power when it comes to processing graph-like objects. However, there is a delicate balance between the increased expressivity and the incurred computational cost.

A significant motivation for studying novel ML algorithms for reconstruction, especially charged particle tracking, is their large computational burden for big data HEP experiments. Figure 1 shows the large increase of expected computational resources needed for all activities in the CMS experiment after the planned major upgrade of the LHC. The largest fraction (60%) of CPU time is consumed by reconstruction-related tasks and of this, the largest component belongs to tracking. The complexity of the current reconstruction algorithms with respect to increasing event density is such that we foresee future shortcomings in computing resources. Several factors contribute to the slowdown in the evolution of single-core CPU performance [47, 48], and highly parallel architectures like graphics processing units (GPUs) now provide more of the computing power in modern high-performance computing centers. While some reconstruction algorithms already take advantage of multithreaded optimizations [49–52], it is a major endeavor to fully migrate the software to highly parallel architectures [53]. Deep learning models offer a natural way to take advantage of GPUs in production. By leveraging greater parallelism, an ML-based algorithm might execute faster with a smaller computational footprint than a traditional counterpart even though it may require more floating point operations (FLOPs).



(a)



(b)

Fig. 1. CPU time annual requirements (in kHEPS06-years) estimated for CMS processing and analysis needs (a) [54, 55]. kHS06-years stands for 10^3 HEP-SPEC06 per year, a standard CPU performance metric for HEP. Two scenarios are considered: one that assumes reaching 275 fb^{-1} per year during Run 4 with 7.5 kHz of data saved and a second that assumes reaching 500 fb^{-1} per year during Run 4 with 10 kHz of data saved (dashed line). The blue curves (and points) show the annual projected CPU need, summed across Tier-0, Tier-1 and Tier-2 resource needs in each of these scenarios. The black curve shows the projected resource availability extrapolating the current CMS processing resources assuming an annual increase of 10–20%. Approximate breakdown of CPU time requirements into primary processing and analysis activities for the first scenario (b) [54, 55].

In this way, the complexity of ML-based algorithms — including the preprocessing and postprocessing steps — may be better than that of existing counterparts.

This chapter is structured as follows. Section 2 provides an overview of the different ways that particle physics data may be encoded as graphs. In Sec. 3, we recapitulate the formalism behind commonly used GNNs. In Sec. 4, we highlight several design considerations, including computational performance, for various approaches to building GNNs for HEP reconstruction. In Sec. 5, we review the suite of GNN applications to tracking and reconstruction tasks. Finally, we summarize the chapter in Sec. 6.

2. Point Cloud and Graph Data

Modern detectors are an assembly of several different technologies with a wide range of spatial granularities (down to $\mathcal{O}(1)$ mm) and a total size of $\mathcal{O}(10)$ m. Therefore, the signals from the detector are extremely heterogeneous. In many cases, the measurements are inherently sparse because of the event configurations of the physics processes. At the same time, the local density of the measurements can be extremely high because of the fine granularity of the active material, for example in the tracker. The signal is also sampled in time, although for most detectors, it is effectively discretized in units of one beam crossing period, which is 25 ns for the LHC.

Locally, a fraction of the data, especially from the calorimeters, can be interpreted as images. In particular, jet images [56] are a now-common representation of localized hadron showers in calorimeters. This has led to proliferation of image-based deep learning techniques, such as CNNs, skip connections, or capsules, for calorimeter- or jet-related tasks with substantial performance improvements over traditional methods [57–62]. However, the image-based representations face some stringent limitations due to the irregular geometry of detectors and the sparsity of the input data. Alternatively, a subset of detector measurements and reconstructed objects can be interpreted as ordered sequences. Methods developed for natural language processing, including RNNs, long-short-term

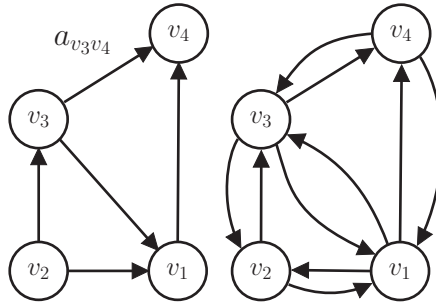


Fig. 2. A directed graph with directed edges (left). If the graph is undirected, it can be transformed into a directed one to obtain a viable input for graph learning methods (right). In particular, each undirected edge is replaced by two directed, opposite edges with identical edge features [19].

memory (LSTM) cells, or gated recurrent units (GRUs), may therefore be applied [63, 64]. While the ordering can usually be justified experimentally or learned [65], it is often arbitrary and constrains how the data is presented to models.

Fundamentally, the raw data is an unordered set of N^v items. However, by additionally considering N^e geometric or physical relationships between items (encoded by an *adjacency matrix*), the set can be augmented into a graph. These relationships may be considered directed or undirected as shown in Fig 2. An adjacency matrix is a (typically sparse) binary $N^v \times N^v$ matrix, whose elements indicate whether a given vertex is adjacent to another vertex. Another, equivalent representation is through an $N^v \times N^e$ *incidence* matrix, whose elements indicate whether a given vertex is connected to a given edge. A third alternative encoding of an adjacency matrix is in coordinate list (COO) format, i.e. a $2 \times N^e$ matrix where each column contains the node indices of each edge. This compact representation is beneficial in terms of incremental matrix construction and reduced size in memory, but for arithmetic operations or slicing a conversion to a compressed sparse row (CSR), compressed sparse column (CSC), or dense format is often necessary.

A graph representation is more flexible and general than images or sequences. In particular, one may recover an image or sequence representation by appropriate choice of the adjacency matrix. Moreover,

there may be less preprocessing required to apply deep learning to this representation of the data. For example, for an image representation of calorimeter hit data, it may be necessary to first cluster the hits, form the two-dimensional energy-weighted image, and center, normalize, rescale, or rotate the image [56, 66]. These manipulations of the data may have undesirable consequences, including loss of particle-level information, distortions of physically meaningful information like jet substructure, modifying Lorentz-invariant properties of the data (e.g. particle mass), and imposing translational invariance in η - ϕ space, which does not respect this symmetry [67]. In contrast, a GNN, may be able to operate on the unclustered hit data, with appropriately chosen connections, directly. Two example HEP detector datasets and their possible graph encoding are illustrated in Fig. 3.

2.1. Graph Construction

In particle physics applications, the specific relationships between set elements to present to an algorithm depends on the context and objective. Subjective choices must be made to construct a graph from the set of inputs. Formally, a graph is represented by a triplet $\mathcal{G} = (\mathbf{u}, V, E)$, consisting of a graph-level, or *global*, feature vector \mathbf{u} , a set of N^v nodes V , and a set of N^e edges E . The nodes are given by $V = \{\mathbf{v}_i\}_{i=1:N^v}$, where \mathbf{v}_i represents the i th node’s attributes. The edges connect pairs of nodes, $E = \{(\mathbf{e}_k, s_k, r_k)\}_{k=1:N^e}$, where \mathbf{e}_k represents the k th edge’s attributes, and s_k and r_k are the vectors of indices of the “sender” and “receiver” nodes, respectively, connected by the k th edge (from the sender to the receiver node). The receiver and sender index vectors are an alternative way of encoding the directed adjacency matrix, as discussed above. The graph and its attributes are represented pictorially in Fig. 4. Edges in the graph serve three different functions:

- (1) the edges are communication channels among the nodes,
- (2) input edge features can encode a relationship between objects, and
- (3) latent edges store relational information learned by the GNN that are relevant for the task.

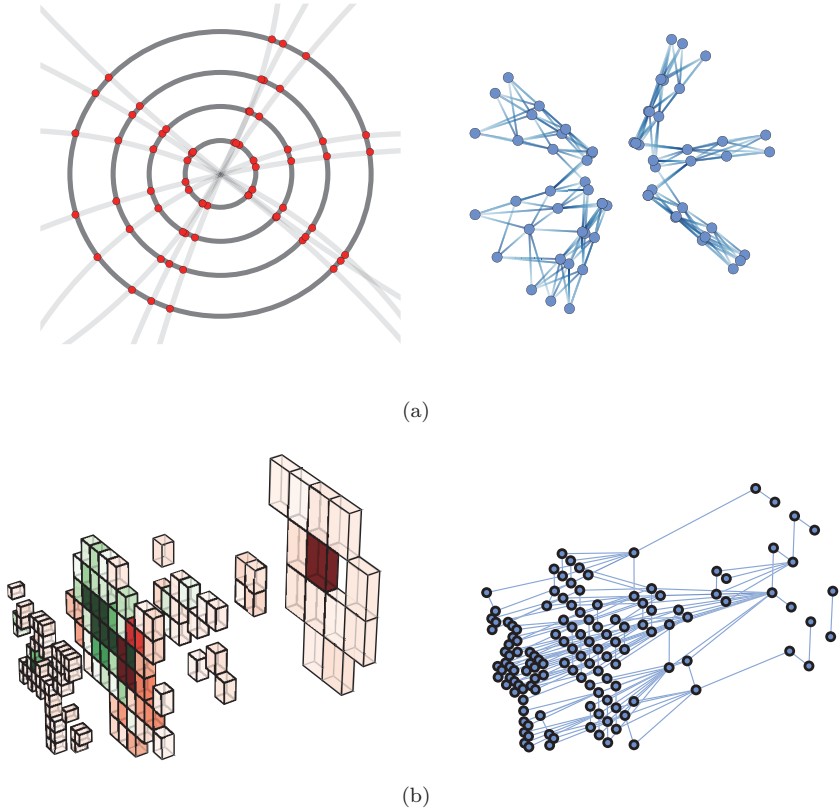


Fig. 3. HEP data lend themselves to graph representations for many applications: segments of hits in a tracking detector hits (a), and neighboring energy deposits in calorimeter cells (b). Figures reproduced from [46].

Depending on the task, creating pairwise relationships between nodes may even be entirely avoided, as in the deep sets [27, 68] architecture with only node and global properties.

For small input sets, with $N^v < 100$, a simple choice is to form a fully-connected graph, allowing the network to learn about all possible object relationships. As the number of edges in a fully-connected graph increases as $N^e \propto (N^v)^2$, the computational cost of applying a neural network to all of the edges becomes prohibitive. A work-around is to precompute a fixed edge feature, such as the geometric distance between nodes, that can be focus on certain neighboring nodes.

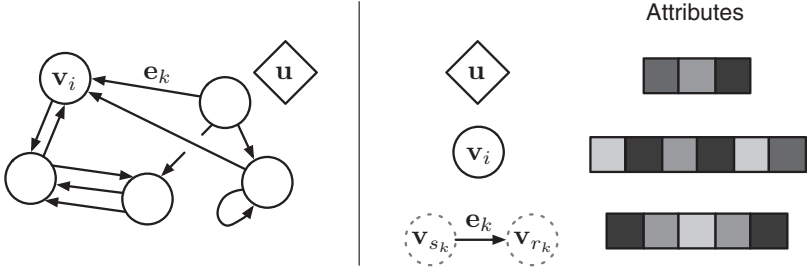


Fig. 4. A directed, attributed multi-graph \mathcal{G} with a global attribute [14]. A node is denoted as \mathbf{v}_i , an edge as \mathbf{e}_k , and the global attributes as \mathbf{u} . The indices s_k and r_k correspond the sender and receiver nodes, respectively, for the one-way edge k (from the sender node to the receiver node).

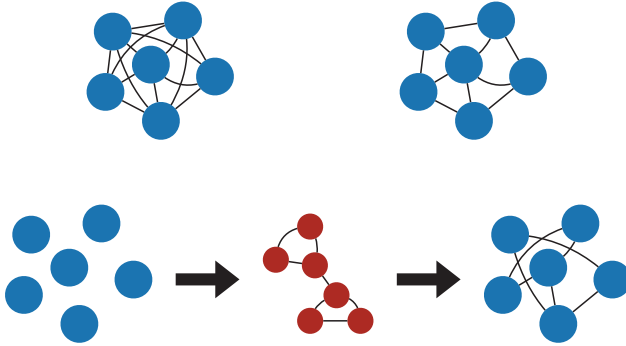


Fig. 5. Different methods for constructing the graph: connecting all pairs of nodes (upper left), connecting neighboring nodes in a predefined feature space (upper right), and connecting neighboring nodes in a latent feature space (lower).

If edge-level computations is required, it may be necessary to restrict the considered edges. Edges can be formed based on the input features (e.g. the $\Delta R = \sqrt{\Delta\phi^2 + \Delta\eta^2}$ between particles) or a learned representation, such as that used by the EdgeConv [13, 30] and GravNet [39] architectures. Given a distance metric between nodes and a criterion for connecting them, such as k -nearest neighbors (kNN) or a fixed maximum distance, the edges can be created. These three different graph construction methods are illustrated in Fig. 5.

3. Graph Neural Networks

GNNs are a class of models for reasoning about explicitly structured data, in particular graphs [8, 11, 12, 69–72]. These approaches all share a capacity for performing computation over discrete entities and the relations between them. Crucially, these methods carry strong relational inductive biases, in the form of specific architectural assumptions, which guide these approaches towards learning about entities and relations [73].

Here, we recapitulate the “graph network” (GN) formalism [14], which synthesizes various GNN methods. Fundamentally, GNs are graph-to-graph mappings, whose output graphs have the same structure as the input graphs. Formally, a GN block contains three “update” functions, ϕ , and three “aggregation” functions, ρ . The stages of processing in a single GN block are:

$$\begin{array}{ll} \text{(Aggregation)} & \text{(Update)} \\ & \mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) \quad \text{(Edge block),} \end{array} \quad (1)$$

$$\bar{\mathbf{e}}'_i = \rho^{e \rightarrow v}(E'_i) \quad \mathbf{v}'_i = \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) \quad \text{(Node block),} \quad (2)$$

$$\begin{array}{ll} \bar{\mathbf{e}}' = \rho^{e \rightarrow u}(E') & \mathbf{u}' = \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) \quad \text{(Global block).} \\ \bar{\mathbf{v}}' = \rho^{v \rightarrow u}(V') & \end{array} \quad (3)$$

where $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$ contains the updated edge features for edges whose receiver node is the i th node, $E' = \bigcup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$ is the set of updated edges, and $V' = \{\mathbf{v}'_i\}_{i=1:N^v}$ is the set of updated nodes. We describe each block below.

The *edge block* computes an output for each edge \mathbf{e}'_k , known as the updated edge feature or “message”. These are subsequently aggregated according to the corresponding receiver nodes $\bar{\mathbf{e}}'_i = \rho^{e \rightarrow v}(E'_i)$ in the first part of the *node block*. These two steps are sometimes known as the graph or edge convolution or message-passing operation. In some ways, this operation generalizes the type of convolution done in CNNs, and the sequential, recurrent processing of RNNs, as shown in Fig. 6. In a 2D convolution, each pixel in an image is processed together with a fixed number of neighboring pixels determined

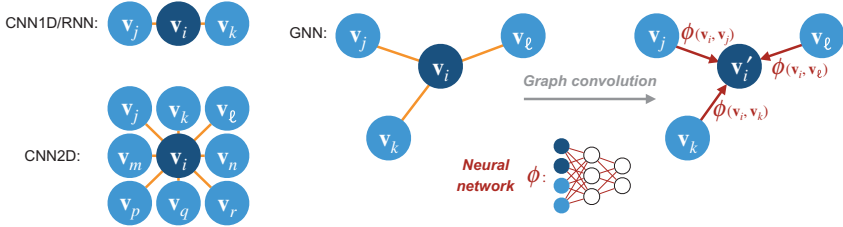


Fig. 6. Input structure for 1D CNNs and RNNs (top left) and 2D CNNs (bottom left) compared to GNNs (right). In a 2D convolution, each pixel in an image can be considered a node with a fixed number of neighbors determined by their proximity and the filter size. RNNs compute sequentially along the input data, generating a sequence of hidden states, as a function of the previous hidden state and the input. A graph convolution operation applies a pair-wise neural network to a variable-size and unordered set of neighboring nodes, and then aggregates the results.

by their spatial proximity and the filter size. RNNs compute sequentially along the input data, generating a sequence of hidden states \vec{h}_t , as a function of the previous hidden state \vec{h}_{t-1} and the input for position t . In contrast, a graph convolution operation applies a pair-wise neural network to all neighboring nodes, and then aggregates the results to compute a new hidden representation for each node \mathbf{v}'_i . As opposed to image and sequence data, the neighbors of a node in a graph are unordered and variable in number.

As described above, the aggregation function $\rho^{e \rightarrow v}$ maps edge-specific information to node-specific outputs by compiling information based on the receiver node indices. To apply generically to unordered graph-structured data, the ρ functions must be invariant to permutations of their inputs, and should take variable numbers of arguments. Examples include an element-wise summation, mean, maximum, and minimum. This construction ensures permutation invariance of the GNN as a whole. In [74], it was shown that this invariance suggests a minimum size for the latent dimension: for scalar inputs the dimensionality of ϕ has to be at least equal to the number of inputs (i.e. nodes or edges) in order to be able to approximate any permutation-invariant function. Other authors have also considered permutation- and group-equivariant constructions [75–82], which are not covered here.

The rest of the node block computes an output for each node $\mathbf{v}'_i = \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$. This can be thought of as an update of the node features, which takes into account the previous node features, the global features, and one round of message passing among neighboring nodes. That is, relational information from nearest neighbors in the graph are used to update the node features.

Finally, the edge- and node-level outputs are each aggregated with $\rho^{e \rightarrow u}$ and $\rho^{v \rightarrow u}$, respectively, in order to compute graph-level information in the *global block*. The output of the GN is the triplet of updated edge, node, and global features, $\mathcal{G}' = (\mathbf{u}', V', E')$ as shown in Fig. 7.

The GN formalism is generic for graph-to-graph mappings. GNs also generalize to graphs not seen during training, because the learning is focused at the edge- and node-level, although such generalization may require conditions to be satisfied between the training and test graph domains [83–85]. Except for the global block, the GN never considers the full graph in a computation. Nonetheless, when multiple GN blocks are stacked in deep or recurrent configurations, information can propagate across the graph’s structure, allowing more complex, long-range relationships to be learned.

As an example of the generality of the GN framework, it can be used to express the dynamic edge convolution (EdgeConv) operation of the dynamic graph CNN (DGCNN) [13], which is commonly used

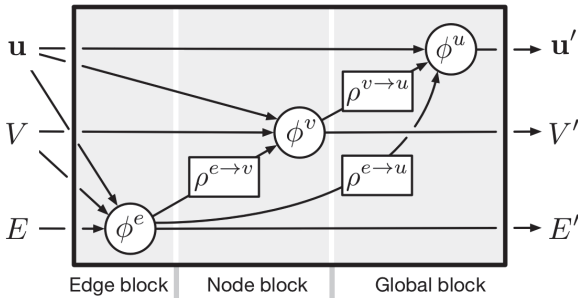


Fig. 7. A GN block from [14] that processes an input graph $G = (\mathbf{u}, V, E)$ and returns a graph with the same structure but updated attributes $G' = (\mathbf{u}', V', E')$.

in HEP. This layer operates on a graph selected using the k -nearest neighbors of the nodes, including self-loops. Edge features are computed as

$$\mathbf{e}'_k = \phi^e(\mathbf{v}_{r_k}, \mathbf{v}_{r_k} - \mathbf{v}_{s_k}). \quad (4)$$

The choice of ϕ^e adopted in [13] is an asymmetric edge function that explicitly combines the global shape structure, captured by the coordinates \mathbf{v}_{r_k} , with local neighborhood information, captured by $\mathbf{v}_{r_k} - \mathbf{v}_{s_k}$. The EdgeConv operation also uses a permutation-invariant aggregation operation $\rho^{e \rightarrow v}$ (e.g. \sum or \max) on the edge features associated with all the edges emanating from each node. The output of the EdgeConv operation at the i th node is thus given by

$$\mathbf{v}'_i = \phi^v(\bar{\mathbf{e}}'_i) = \bar{\mathbf{e}}'_i, \quad (5)$$

that is the ϕ^v function is trivial. A crucial difference with the GN framework is that after each EdgeConv layer, the connectivity of the graph is recomputed using the k -nearest neighbors in the latent space. This dynamic graph update is the reason for the name of the architecture. Similarly, GravNet and GarNet [39] are two other GNN architectures that use the distance in a latent space when aggregating to predict a new set of node features.

Other GNN models are also expressible within this framework or with minor modifications. For instance, interaction networks [9] use a full GN block except for the absence of the global features to update the edge properties. Deep sets [68] bypass the edge update completely and predict the global output from pooled node information directly. PointNet [10] use similar update rule, with a max-aggregation for $\rho^{v \rightarrow u}$ and a two-step node update.

Another class of models closely related to GNNs that perform predictions on structured data, especially sequences, are *transformers*, based on the self-attention mechanism [86]. At a high level, a self-attention layer is a mapping from an input sequence, represented as a $n \times d_{\text{in}}$ matrix X (where n is the sequence length and d_{in} is the dimensionality of the input features) to a $n \times d_{\text{out}}$ output matrix through an attention function, which focuses on certain positions of the input sequence. A self-attention function takes as input an $n \times d_k$ query

matrix Q , and a set of key-value pairs, represented by a $n \times d_k$ matrix K and a $n \times d_v$ matrix V , respectively, all of which are transformed versions of the input sequence

$$Q = XW_Q, K = XW_K, V = XW_V, \quad (6)$$

where W_Q , W_K , and W_V are learnable $d_{\text{in}} \times d_k$, $d_{\text{in}} \times d_k$, and $d_{\text{in}} \times d_{\text{out}}$ matrices, respectively. The scaled dot-product attention (see Fig. 8) is computed by taking the dot products of the query with all keys (as a compatibility test) divided by $\sqrt{d_k}$ and applying a softmax function to obtain the weights for the values. In matrix form:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \quad (7)$$

An important variant of this is *multi-head attention* depicted in Fig. 8: instead of applying a single attention function, it is beneficial to project the queries, keys, and values h times into subspaces whose dimensions are h times smaller. On each of these projected versions of queries, keys, and values, the attention function is computed yielding h d_v -dimensional output values. These are concatenated and

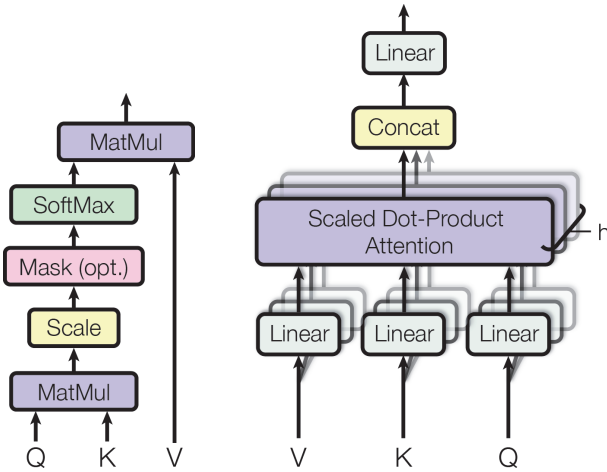


Fig. 8. Scaled dot-product attention (left) and multi-head attention (right), consisting of several attention layers running in parallel, from [86].

once again projected, resulting in the final values:

$$\text{MultiHead}(X) = \text{concat}_{i \in [h]} [H^{(i)}] W^O \quad (8)$$

$$\text{where } H^{(i)} = \text{Attention}(XW_Q^{(i)}, XW_K^{(i)}, XW_V^{(i)}), \quad (9)$$

and W^O is a learnable $hd_v \times d_{\text{out}}$ matrix. In practice, a simplifying choice of $d_{\text{in}} = hd_k = hd_v = d_{\text{out}}$ is typically made. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

In the language of GNNs, a transformer computes normalized edge weights in a fully-connected graph, and passes messages along the edges that are aggregated in proportion to these weights. For example, the transformer in the graph attention network [87] uses a ϕ^e function that produces both a vector message and an unnormalized weight. The aggregator $\rho^{e \rightarrow v}$ then normalizes the weights before computing a weighted sum of the message vectors. This allows the edge structure among the input nodes to be inferred and used for message passing. In addition, attention mechanisms are a way to apply different weights in the aggregation operations ρ .

Another extension of GNNs involves graph pooling, represented in Fig. 9. Graph pooling layers play the role of “downsampling”, which coarsens a graph into a sub-structure. Graph pooling is mainly used for three purposes: to discover important communities in the graph, to imbue this knowledge in the learned representations, and to reduce the computational costs of message passing in large scale

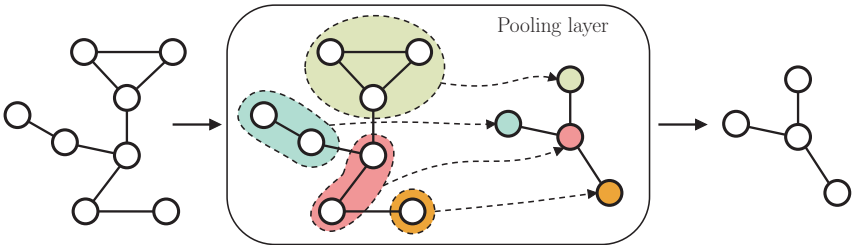


Fig. 9. An example of a graph pooling layer that coarsens the graph by identifying and clustering nodes of the same neighborhood together, so that each group becomes a node of the coarsened graph [19].

structures. Pooling mechanisms fall in two broad classes: adaptive and topological.

Adaptive graph pooling relies on a parametric, trainable pooling mechanism. One example of this approach is differentiable pooling [88], which uses a neural network layer to learn a clustering of the current nodes based on their embeddings at the previous layer. Top- k pooling [89] learns node scores and retain only the entries corresponding to the top nodes. Node selection is made differentiable by means of a gating mechanism built on the projection scores. Self-attention graph (SAG) pooling [90] extends top- k pooling by using a GNN to learn attention scores. Another example is edge pooling [91], in which edge scores are computed and edges are contracted iteratively according to those scores. In contrast to these adaptive methods, topological pooling mechanisms are not required to be differentiable and typically leverage the structure of the graph itself. The graph clustering software (GRACCLUS) [92] implements a widely-used, efficient greedy clustering algorithm that matches vertices based on their edge weights. Similarly, nonnegative matrix factorization pooling [93] provides a soft node clustering using a nonnegative factorization of the adjacency matrix.

4. GNN Design Considerations

The formalism and methods introduced in Sec. 3 expose the numerous dimensions of the space of GNN model architectures. While the possibilities for combining the ingredients of GNN are limitless, other considerations and constraints come into play to shape the model for a given task and environment. In this section, we discuss some of the salient facets of GNN design for HEP reconstruction tasks. These are some of the guiding principles that lead to the models used for the applications we describe further in Sec. 5.

4.1. *Model architectures*

Many of the choices in the design the GNN model architectures reflect the learning objectives or aspects of the data that are specific to HEP.

The choice of architecture is an important way to incorporate inductive bias into the learning task. For instance, this choice includes the size of the networks, the number of stacked GNN blocks, attention mechanisms, and different types of pooling or aggregation. The model architecture should reflect a logical combination of the inputs towards the learning task. In the GN formalism, this means a concrete implementation of the block update and aggregation functions and their sequence. As an example of such a choice, global aggregation can occur before a node update, or an edge representation can be created and aggregated to form a node update. The difference between the two is that one is based on a sum of pairwise representations, and the other on a global sum of node representations.

Stacks of GN blocks are also useful for two purposes. First, just as in CNNs, they can construct a higher-level, more abstract representation of the data. Second, the number of iterations of message passing defines the nodes that can exchange information. This is illustrated in Fig. 10. Multiple iterations increase each nodes' neighborhood of communication, as the representation of its neighboring nodes was previously updated with messages from their neighbors.

Attention mechanisms also play an important role in emphasizing or deemphasizing certain nodes or connections during aggregation. A popular choice is to use the ΔR distance between measurement nodes in the input space or Euclidean distance in the latent space (or subspace) as an edge weight. Others networks [20] use the network's predicted edge weight, which acts to reinforce its learned connections. Finally, the choice of aggregation method is crucial to keep open the appropriate communication channels and maintain the desired properties of the output, such as permutation invariance.

4.2. *Graph reduction and alternative loss functions*

One difficulty of applying deep learning to HEP data is the “jagged” or event-dependent nature of the target. In particular, the number of physics objects, such as tracks, clusters, or final-state particles, to be reconstructed per event is variable and unknown *a priori*. For this

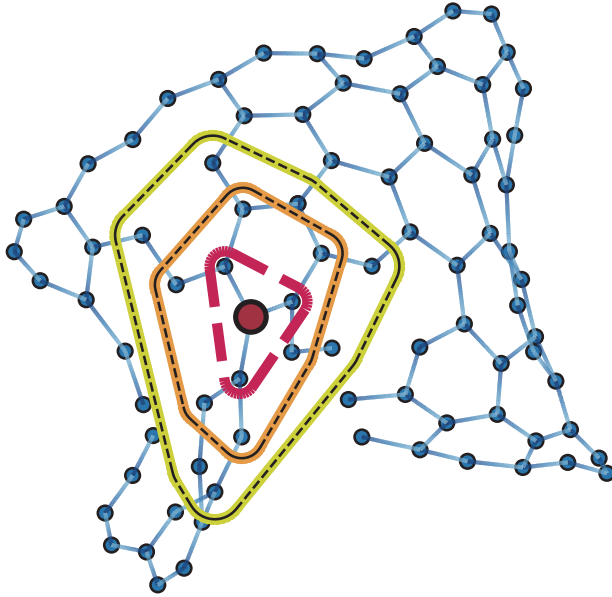


Fig. 10. The red, orange-highlighted, and yellow-highlighted dotted lines represent the enlarging neighborhood of nodes that may communicate with the red node after one, two, and three iterations of message passing, respectively [46]. Those nodes outside of the yellow-highlighted dotted boundary do not influence the red node after three iterations.

reason, methods based on a fixed output size for the output are challenging to apply.

Two methods [42, 94] aim to specifically address this problem. In [42], a clustering or “condensation” of the input nodes is derived through a choice of condensation points and a dual prediction of a regression target and a condensation weight. The loss function is inspired by attractive and repulsive electromagnetic potentials, ensuring that nodes that belong to the same target object are kept close in the latent space. Similarly, a dynamic reduction network is proposed in [94] uses a DGCNN [13] and a greedy popularity-based clustering algorithm [95] to reduce the number of nodes. The model was developed for reconstructing HEP data from granular calorimeters, although currently results are only presented for the MNIST superpixel dataset [96].

Another aspect to consider is whether the loss function construction preserves the symmetries of GNN algorithm when predicting unordered sets. For instance, traditional loss functions like the mean-squared error (MSE) are not invariant with respect to permutations of the output and target sets because the outputs must be reconstructed in the same order as the targets to achieve a small value of the loss function. To preserve this property, alternative permutation-invariant loss functions like the Chamfer distance [97–99], Hungarian loss [100], and differentiable approximations of the Earth mover’s distance [98, 101, 102] have been proposed.

4.3. Computational performance

One of the most crucial factors in determining the computational performance of a GNN is the graph connectivity. The number of edges in a graph usually defines the memory and speed bottleneck, because there are typically more edges than nodes and the ϕ^e function is applied the most times. If the graph is densely connected, the number of edges scales quadratically with the number of nodes $N^e \propto (N^v)^2$. Even without such as severe scaling, if the ϕ^e is a large neural network or if there are multiple stacked blocks, the computational resources needed can still be large. For instance, the tracking GNN of [21] takes as input a portion of a collision event containing approximately 2500 nodes and 25,000 edges. Given the size of the networks and the multiple repeated iterations, one inference requires 52 GFLOPs. As such, it is imperative to study effective pruning and network compression techniques [103–108], reduced precision [109–111], and alternative hybrid network architectures [112–114] designed to be more efficient.

Another consideration for building and efficiently training GNNs on hardware is whether to use dense or sparse implementations of the graph’s adjacency matrix. A dense adjacency matrix supports fast, parallel matrix multiplication to compute E' , which, for example, is exploited in GCNs and transformers. However, the adjacency matrix’s memory footprint is quadratic in the number of nodes: 10,000 fully-connected nodes corresponds to an adjacency matrix with 100,000,000 entries and thus 400 MB for a 32-bit representation

or 12.5 MB with a binary representation. Alternatively, using sparse adjacency matrices implies the memory scales linearly in the number of edges, which allows much larger graphs to be processed. However, the sparse indexing operations required to implement sparse matrix multiplication can incur greater computational costs than their dense counterparts. Such sparse operations are a bottleneck in current deep learning hardware, and next-generation hardware may substantially improve their speed, this would potentially improve the relative advantage of sparse edge implementations of GNNs.

An important advantage of GNN-based approaches over traditional methods for HEP reconstruction is the ability to natively run on highly parallel computing architectures. All of the deep learning software frameworks for graphs, like PyTorch Geometric [115], Deep Graph Library [116], DeepMind’s `graph_nets` [117] and `jraph` [118] libraries, StellarGraph [119], and Spektral [120, 121], support GPUs to parallelize the algorithm execution. Work has also been done to accelerate the inference of deep neural networks with field-programmable gate arrays (FPGAs) [109–111, 122–127], including GNNs [128, 129], and using heterogeneous computing resources as a service [130–132]. Graph processing on FPGAs, reviewed in [133], is a potentially promising direction. However, we note that detailed and fair comparisons of the computational and physics performance between GNN-based algorithms and traditional HEP algorithms have not yet been extensively performed. This is a major deliverable of future work.

5. Applications to Particle Physics Tasks

In this section, we review applications of graph neural networks to a variety of reconstruction tasks in high-energy physics. The main graph learning objectives used in HEP reconstruction tasks are

- *edge classification*: the prediction of edge-level outputs used to classify edges,
- *node classification or regression*: the prediction of node-level outputs, representing class probabilities or node properties,

- *graph pooling*: associating related nodes and edges and possibly predicting properties of these neighborhoods, and
- *global graph classification*: prediction of a single vector of probabilities the entire graph; this is common for jet and event identification at the LHC and neutrino event classification, but not covered here.

5.1. *Charged particle tracking*

In HEP data analysis, it is crucial to estimate the kinematics of the particles produced in a collision event, such as the position, direction, and momentum of the particles at their production points, as accurately as possible. For this purpose, a set of tracking devices (or *trackers*) providing high-precision position measurements is placed close to the beam collision area. Charged particles created in the collisions ionize the material of these devices as they exit the collision area, providing several position measurements along the trajectory of each particle. To prevent the detector elements from disturbing the trajectory of the particles, the amount of material present in such tracking detectors is kept to a minimum. The tracker is usually immersed in a strong magnetic field that bends the trajectory, as a means to measure the components of the momentum — the curvature is proportional to the momentum component transverse to the magnetic field.

The task of track reconstruction is traditionally divided into two subtasks, track finding and track fitting, although modern techniques may combine them [134, 135]. Track finding is a pattern recognition or classification problem and aims at dividing the set of measurements in a tracking detector into subsets (or track candidates) containing measurements believed to originate from the same particle. An illustration of a simple track finding problem is shown in Fig. 11. It is the task of track finding to associate hits to their respective tracks.

The track fit takes the set of measurements in a track candidate and estimates as accurately as possible a set of parameters describing the state of the particle somewhere in the tracking detector, often at a reference surface close to the particle beam. The fitted parameters

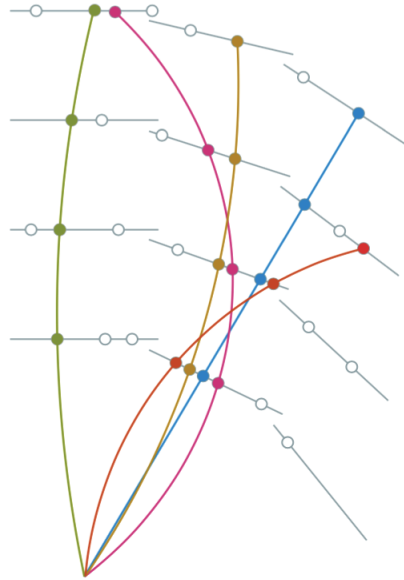


Fig. 11. Illustration of the charged particle tracking task [134]. Each colored curve is the trajectory of a charged particle in a constant magnetic field perpendicular to the viewing plane. The solid circles are hits left by the particle as it traverses the sensitive layers. Empty circles are spurious noise hits not created by a reconstructible particle.

of the track, especially the curvature, allow for the measurement of the momentum and charge of the particle. Ideally, each particle would leave one and only one hit on each layer of the detector, the trajectories would be exact helices, and the coordinates would be exact. In reality, particles may leave multiple hits or no hits in a layer, inhomogeneities in the magnetic field result in distorted arcs, particles may undergo multiple scattering, and the measurements may have anisotropic uncertainties. Given that these complications are commonplace, a solution that is robust to them is desirable.

Current tracking algorithms include the combinatorial track finder (CTF) [136, 137] based on the combinatorial Kalman filter [138–141] that allows pattern recognition and track fitting to occur in the same framework. Another tracking algorithm uses a Hough transform [142] to identify groups of hits that are roughly consistent with

a track hypothesis, reducing the combinatorial background in the downstream steps. This algorithm is optimized for the real-time trigger system. One major computational bottleneck common to many of these algorithms is the combinatorial explosion of possible track candidates, or *seeds*, in high hit density environments. Improved track seeding, based on global pattern recognition, can dramatically improve the computational performance [143].

Lately, there has been increased interest in exploring new methods to address the trade-off between algorithmic quality (good track reconstruction) and speed, which motivated the TrackML particle tracking challenge (see Chapter 20) [134]. From the ML point of view, the problem can be treated as a latent variable problem similar to clustering, in which particle trajectory “memberships” must be inferred, a sequence prediction problem (considering trajectories as time series), a pattern denoising problem treating the sampled trajectories as noisy versions of ideal, continuous traces, or an edge classification problem on graph-encoded hit data.

The authors of [20] propose a GNN approach to charged particle tracking using edge classification. Each node of the graph represents one hit with edges constructed between pairs of hits on adjacent tracker layers that may plausibly belong to the same track. After multiple updates of the node representation and edge weights and using the learned edge weight as an attention mechanism, the “segment classifier” model learns which edges truly connect hits belonging to the same track. This approach transforms the clustering problem into an edge classification by targeting the subgraphs of hits belonging to the same trajectories. This method has high accuracy when applied to a simplified scenario, and is promising for more realistic ones. In [21] from the same authors, an updated GNN model, based on stacked, repeated interaction network [9] layers, is presented and provides improved performance. Figure 12 shows the updated architecture, in which the same interaction network layer operates on the initial latent features H_0 concatenated with the current features H_{i-1} . After eight iterations, the output FC network takes the last latent features H_8 to produce classification scores for every edge. Figure 13 shows the performance of the GNN in correctly classifying

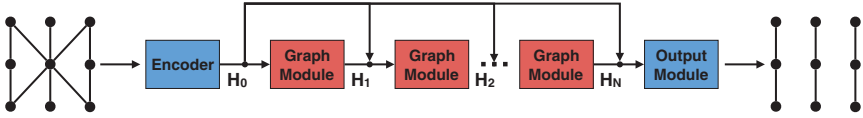


Fig. 12. Graph neural network architecture for particle tracking [21]. The initial latent features of the nodes and edges after the encoder network are named H_0 . The graph module is applied repeatedly to the latent features. For the i th iteration, the initial features H_0 are concatenated with the current features H_{i-1} . After eight iterations, the output network takes the last latent features H_8 to produce classification scores for every edge.

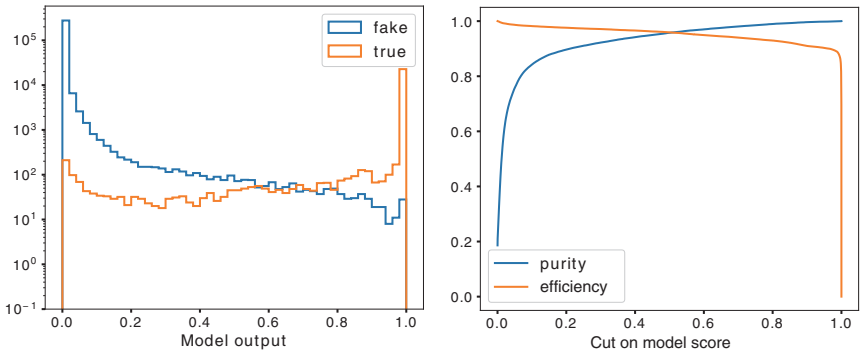


Fig. 13. The distribution of the segment classifier scores predicted by the GNN from [21] for true segments (orange) and fake segments (blue), showing clear separation between the two (left). The track segment purity (blue) and efficiency (orange) as a function of different cuts on the model score (right). With a threshold of 0.5 on the GNN output, the edge efficiency, defined as the ratio of the number of true edges passing the threshold over the number of total true edges, reaches 95.9%, and the purity, defined as the ratio of the number of true edges passing the threshold over the number of total edges passing the threshold, is 95.7%.

the edges, which reaches 95.9% efficiency and 95.7% purity on the simulated TrackML dataset [134] consisting of top quark–antiquark pairs produced with an additional 200 pileup interactions overlaid to simulate the expected conditions at the HL-LHC.

Reference [22] presents further extensions to this model with preprocessing and postprocessing steps. In particular, the authors explore constructing graphs from learned representations that contain a nonlinear metric structure, which allows for efficient clustering

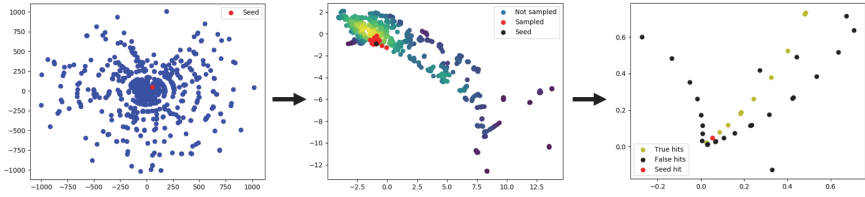


Fig. 14. A single seed hit (red) and its neighborhood (blue) in the x - y plane in one event are shown (left). Using the graph embedding method of [22], hits (red) that fall within radius ϵ of the seed hit (black) are shown in a 2D projection of the embedded space (center). Hits in the embedded space neighborhood are shown projected back into the original space (right).

and neighborhood queries of data points. An FC model ψ , parameterized by weights θ and mapping input hits x into a new Euclidean space $\psi(x|\theta) \in \mathbb{R}^d$, is trained using a hinge embedding loss, pulling together points belonging to the same particle and pushing apart points that do not. Figure 14 shows the process by which neighboring hits are selected in the embedded space. The embedded graphs are then fed into doublet, triplet, and end-to-end track classifiers by clustering in the embedded space. A set of postprocessing methods is also used to improve the performance with knowledge of the detector physics. Considering the central barrel region of the detector ($-2 < \eta < 2$), they demonstrate a seed efficiency greater than 93%, purity greater than 99%, and a track finding TrackML score of 0.932 (given reconstructability constraints), which compare favorably with traditional methods while allowing for greater parallelizability.

Reference [23] applies similarity hashing using approximate nearest neighbors [24, 25] to identify “buckets” of hits, with clustering in a latent space through a custom loss function to group hits into tracks within that bucket. This approach treats charged particle tracking as a clustering problem and attempts to find the ideal feature space in which clusters (hits belonging to the same track) are isolated enough for a distance threshold to split them, yet compact enough for the clusters to not be split themselves. The custom loss function balances these objectives using three terms: the first is proportional to the variance within each cluster (attracts hits within the same cluster), the second is inversely proportional to the variance of different

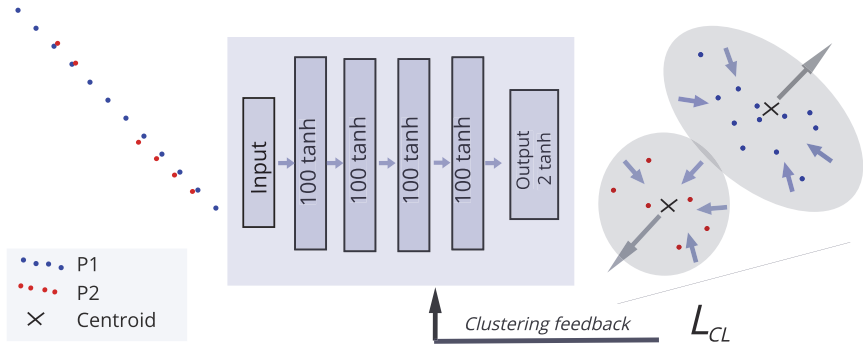


Fig. 15. Similarity hashing with clustering in a latent space to group hits into tracks [23]. The model takes a bucket from the approximate nearest neighbors search as input and maps it into a hidden feature space where the clusters representing different particles are well-separated after training with a custom loss function.

clusters' centroids (repels hits within different clusters), and the third rewards more compact clusters. Finally, once the model is trained, a simple agglomerative clustering procedure can be used in the latent space to group hits belonging to the same tracks. This approach is shown in Fig. 15 and connects to approaches discussed in Sec. 5.5.

5.2. Secondary vertex reconstruction

The particles that constitute a jet often originate from various intermediate particles that are important to identify in order to fully characterize the jet. The decay point of the intermediate particle can be identified as a *secondary vertex* (SV), using clustering algorithms on the reconstructed tracks, such as adaptive vertex reconstruction [144–146], the CMS inclusive vertex finder [147], or the ATLAS SV finder [148]. A review of classical and adaptive algorithms for vertex reconstruction can be found in [135].

Based on the association to a SV, the particles within a jet can be partitioned. Properties of the secondary vertices, such as flight distance and total associated energy and mass may then be used in downstream algorithms to identify jets from the decay of bottom or charm quarks.

Through the lens of GNNs, SV reconstruction can be recast as a edge classification and graph partitioning problem. In [40], the authors develop a general formalism for set-to-graph (Set2Graph) deep learning and provide mathematical proof that their model formulation is a universal approximator of set-to-graph functions. In particular, they apply a set-to-edge approximation to the problem of SV reconstruction (particle association) within a jet. The target is to classify each edge based on whether the two associated particles originate from the same vertex. The model composes an embedding, a fixed broadcasting map, and a graph-to-graph model to produce the final edge scores. Though built from simple components, the model’s expressivity stems from the equivariant formulation. Their model outperforms other ML methods, including a GNN [149], a Siamese network [150–152], and a simple multilayer perceptron, on the jet partitioning task by about 10% in multiple metrics.

Reference [41] extends this work and demonstrates the SV reconstruction performance for bottom, charm, and light quark jets, separately, in simulated top quark–antiquark pair events. In almost all cases, the Set2Graph model outperforms the standard adaptive vertex reconstruction (AVR) algorithm [135, 153], and a simpler, less expressive Set2Graph model called the track pair (TP) classifier. Figure 16 shows the Set2Graph model architecture. The performance may be quantified in terms of the adjusted Rand index (ARI) [154], which measures the fraction of correctly assigned edges normalized to the expected fraction from random clustering. They observe a large improvement (33–100%) in mean ARI for bottom and charm quark jets, and a slight improvement (1%) for light jets over the AVR and TP classifiers.

5.3. *Pileup mitigation*

To increase the likelihood of producing rare processes and exotic events, the transverse size of the colliding beams can be squeezed, resulting in multiple interactions per beam crossing. The downside of this increased probability is that, when an interesting interaction occurs, it is accompanied by simultaneous spurious interactions

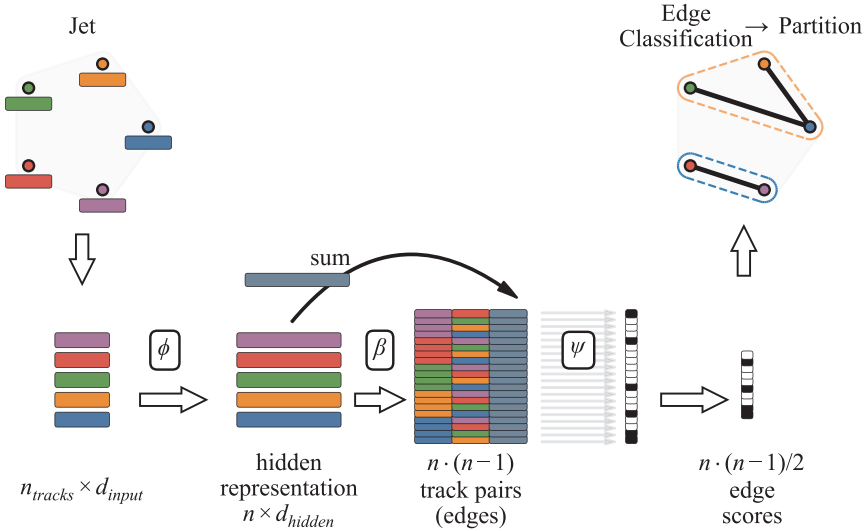


Fig. 16. The Set2Graph [40, 41] model architecture (top) consisting of a set-to-set component ϕ , which creates a hidden representation of each track, a broadcasting layer β , which creates a representation for each directed edge (ordered pair of tracks in the jet), and an edge classifier ψ . Edges whose symmetrized edge score is over a certain threshold are connected, resulting in the set partition.

(called *pileup*), considered as noise for the analysis. For instance, the rate of simultaneous interactions per bunch crossing is projected to reach an average of 140–200 for the high-luminosity LHC and 1000 for the proposed 100 TeV hadronic Future Circular Collider (FCC-hh) [155]. Pileup increases the likelihood of error in the reconstruction of events of interest because of the contamination from particles produced in different pileup interactions. Mitigation of pileup is of prime importance to maintain good efficiency and resolution for the physics objects originating from the primary interaction. While it is straightforward to suppress charged particles from pileup by identifying their origin, neutral particles are more difficult to suppress. One of the current state-of-the-art methods is to compute a pileup probability weight per particle [156] using the local distribution shape, and to use it when computing higher-level quantities. As a graph-based task, this can generally be conceptualized as a node classification problem.

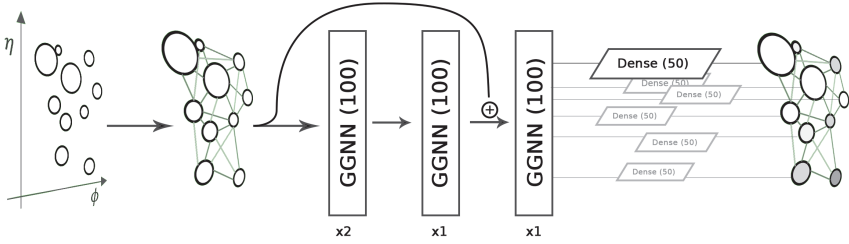


Fig. 17. Gated graph network architecture used for pileup mitigation in [44]. The event is preprocessed by linking local particles together, after which it is fed to 3 gated GNN layers with time steps 2, 1, and 1, respectively, including a residual connection from the first to the third layer. Then an FC network calculates a pileup classification score individually for each graph node.

In [44], the authors utilize the gated GNN architecture [157], shown in Fig. 17, to predict a per particle probability of originating from the pileup interactions. The graph comprises one node per charged and neutral particle of the event, and the edge connectivity is restricted geometrically to $\Delta R < 0.3$ in the η - ϕ plane. The per-particle pileup probability is extracted with an FC model after three stacked graph layers and a skip connection into the last graph layer. The model outperforms other methods for pileup subtraction, including GRU and FC network architectures, and improves the resolution of several physical observable.

The authors of [45] take inspiration from the graph attention network [87] and the graph attention pooling network (GAPNet) [158] to predict a per-particle pileup probability with a model called attention-based cloud network (ABCNet) shown in Fig. 18. The node and edge features are updated by multiple FC models, where each (directed) edge is weighted by an attention factor. The connectivity is initialized to the k -nearest neighbors in the feature space then updated based on the latent space of the stacked graph layers. A multi-head attention mechanism, described in Sec. 3, is used to improve the robustness of models. Skip connections further facilitate the information flow. A global graph latent representation is used to compute an output for each node using a fixed ordering. This method improves the resolution of the single jet and dijet mass observables over a large range of number of pileup interactions.

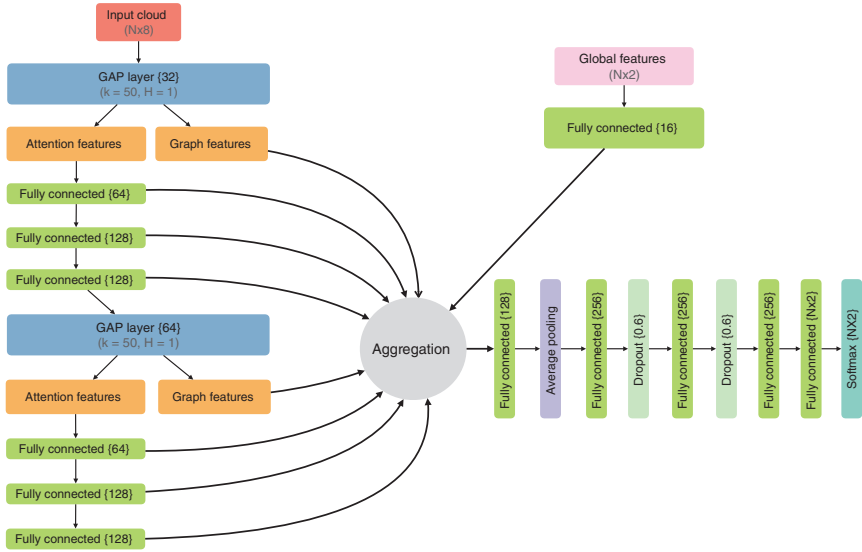


Fig. 18. ABCNet architecture used for pileup identification in [18]. FC layer and encoding node sizes are denoted inside curly brackets. For each graph attention pooling layer (GAPLayer), the number of k -nearest neighbors and attention heads (h) are given.

5.4. Calorimeter reconstruction

A *calorimeter* is a block of instrumented material in which particles to be measured are fully absorbed and their energy transformed into a measurable quantity. Typically, the interaction of the incident particle with the detector produces a cascade of secondary particles (known as a *shower*) with progressively smaller energies. The energy deposited by the showering particles in the calorimeter can be detected in the form of charge or light and serves as a measurement of the energy of the incident particle. There are two primary categories of particle showers, one caused by the electromagnetic force and consisting of electrons, positrons, and photons, and the other resulting from the strong nuclear force and composed of charged and neutral hadrons. Corresponding to these two types of particle showers, there are the two primary forms of calorimeters: electromagnetic and hadron calorimeters.

Calorimeters can be further classified into sampling and homogeneous calorimeters. Sampling calorimeters consist of alternating layers of an *absorber*, a dense material used to induce the shower and energy loss of the incident particle, and an active medium that provides the detectable signal. Conversely, homogeneous calorimeters are built of one type of material that performs both tasks, energy degradation and signal generation. Nonetheless, both types are usually segmented into different cells, providing some spatial resolution. Moreover, reconstruction of the energy of the incoming particle in a calorimeter requires joint clustering and calibration of the signal in various cells. Reviews of classical techniques for calorimetry in high-energy physics can be found in [159–161]. From a GNN perspective, calorimeter reconstruction can be thought of as (possible) graph pooling and node regression.

Reference [39] proposes a GNN-based approach to cluster and assign signals in a high granularity calorimeter to separate particles. A latent edge representation is constructed using a potential function of the Euclidean distance d_{jk} between nodes j and k in (a subspace of) the latent space

$$V_n(d_{jk}) = \exp(-|d_{jk}|^n) \quad (10)$$

as an attention weight. One proposed model — GravNet — connects the nearest neighbors in a latent space and uses the potential V_2 , while another — GarNet — uses a fixed number of additional nodes to define the graph connectivity and V_1 as the potential. Node features are updated using the concatenated messages from multiple aggregations, and the output predicts the fraction of a cell’s energy belonging to each particle. These methods improve over classical approaches and could be more beneficial in future detectors with greater complexity.

Reference [21] also proposes a GNN approach using stacked Edge-Conv layers to identify clusters in the CMS high granularity calorimeter. The output is a set of edge weights classifying hit pairs as being particles or noise. Results are promising in that muons, photons, and pions are efficiently and purely reconstructed and their energy is accurately measured as shown in Fig. 19 in the case of photons.

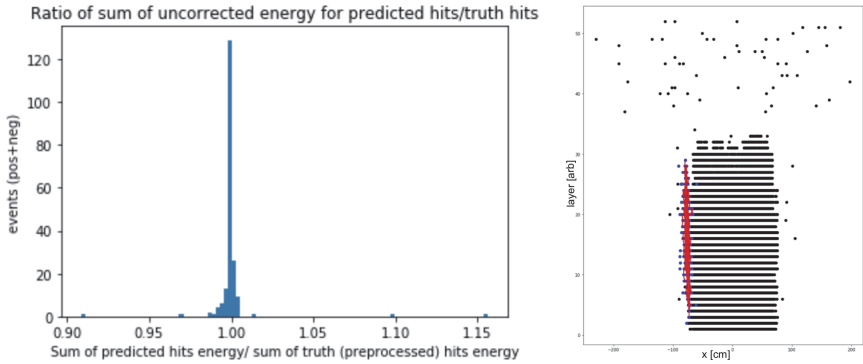


Fig. 19. The ratio, per event, for photons of total collected calorimeter energy deposits connected by predicted edges to the energy collected by the associations from ground truth (left) for a GNN from [21]. The event display of a single photon showing the predicted edges in red, the truth nodes in blue, and the energy deposits from noise in black (right).

Ongoing work includes studies on how to reconstruct multiple particle types simultaneously using network architectures that can assign categories to edges, and how to deal with overlapping showers and fractional assignment of hit energy into clusters.

5.5. Particle-flow reconstruction

Modern general-purpose detectors at high-energy colliders are composed of different types of detector layers nested around the beam axis in addition to forward and backward “endcap” layers. Charged particle tracks are measured by a tracking detector as described in Sec. 5.1. As described in Sec. 5.4, electrons and photons are absorbed in an electromagnetic calorimeter (ECAL), creating *clusters* of energy that can be measured. Similarly, charged and neutral hadrons are absorbed, clustered, and measured in a hadron calorimeter (HCAL). Muons may produce hits in additional tracking layers called muon detectors, located outside of the calorimeters, while neutrinos escape unseen. Figure 20 displays a sketch of a transverse slice of a modern general-purpose detector, the CMS detector [162] at the CERN Large Hadron Collider (LHC), with different types of particles and their corresponding signatures.

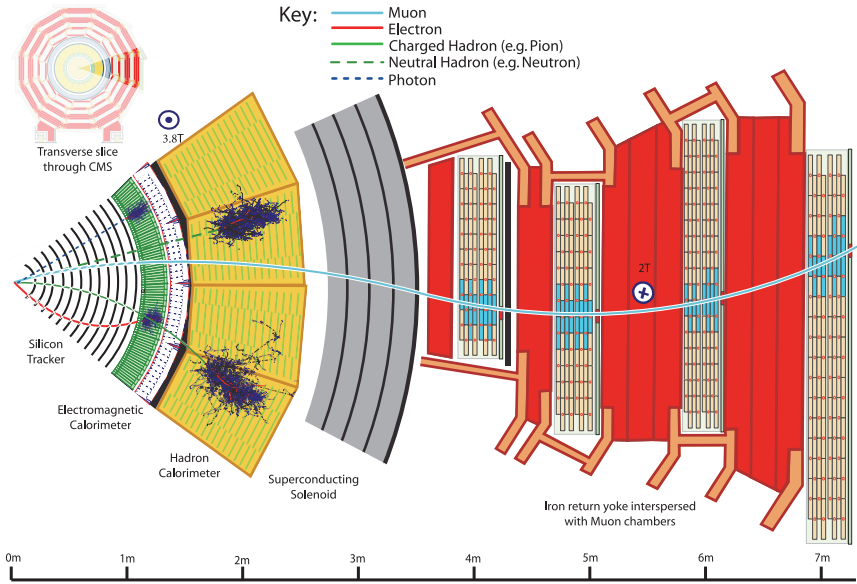


Fig. 20. Different types of particles and their corresponding signatures in the CMS detector [163]. Particle-flow algorithms aim to optimally combine different measurements for different subdetectors to reconstruct a list of final-state particles.

An improved global event description can be achieved by correlating the basic *elements* from all detector layers (tracks and clusters) to identify each final-state particle, and by combining the corresponding measurements to reconstruct the particle properties. This holistic approach is called *particle-flow (PF) reconstruction*. The PF concept was developed and used for the first time by the ALEPH experiment at LEP [164] and has been successfully deployed at the LHC in both CMS [163] and ATLAS [165]. An important ingredient in this approach is the fine spatial granularity of the detector layers. The ultimate goal of PF reconstruction is to provide a complete list of identified final-state particles, with their momenta optimally reconstructed from a combined fit of all pertaining measurements, and links to contributing elements. From this list of particles, the physics objects can then be determined with superior efficiencies and resolutions. This is shown schematically in Fig. 21.

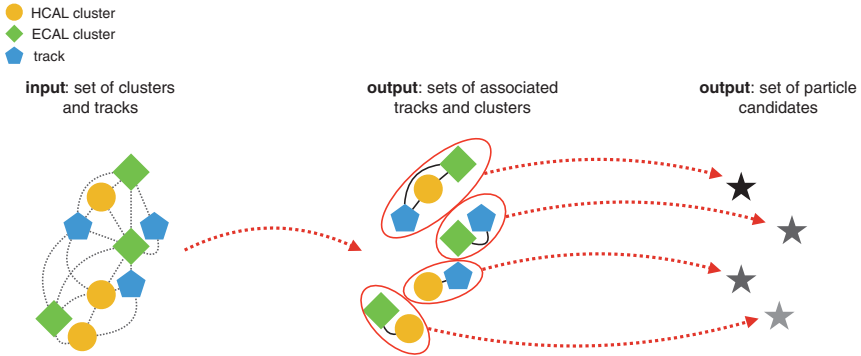


Fig. 21. Schematic representation of a particle-flow algorithm based on input HCAL cluster, ECAL clusters, and tracks. These inputs are associated to one another and the list of final-state particles is determined by combining these measurements.

ML methods based on an image representations have been studied for PF reconstruction. Based on a computer-vision approach, [166] uses a CNN with up and down sampling via choice of kernel size and stride to combine information from ECAL and HCAL layers to better reconstructed the energies of hadron showers. As a graph-based learning problem, PF reconstruction has multiple objectives: graph pooling or edge classification for associating input measurements to output particles and node regression for measuring particle momenta.

Reference [42] proposes the *object condensation* loss formulation using GNN methods to extract the particle information from the graph of measurements as well as grouping of the measurements. The model predicts the properties of a smaller number of particles than there are measurements, in essence reducing the graph without explicit assumptions on the number of targeted particles. Certain nodes are chosen to be the “condensation” point of a particle, to which the target properties are attached. A stacked GravNet model performs node-level regression of a kinematic correction factor together with a *condensation weight* β_i , which indicates whether that node is representative of a particle in the event. A special loss function mimics attractive and repulsive electromagnetic potentials

to ensure nodes belonging to the same particle are close in the latent space. Explicitly, an effective charge is computed from the condensation weight through a function with zero gradient at 0 and monotonically increasing gradient towards a pole at 1: $q_i = \text{arctanh}^2 \beta_i + q_{\min}$. The node α with maximum charge q_α for each particle is used to define an attractive potential $\check{V}_k(x) = \|x - x_\alpha\|^2 q_{\alpha k}$ or a repulsive potential $\hat{V}_k(x) = \max(0, 1 - \|x - x_\alpha\|) q_{\alpha k}$ depending on if the node α belongs to the same particle. This is combined in the loss function,

$$L_V = \frac{1}{N} \sum_{j=1}^N q_j \sum_{k=1}^K \left(M_{jk} \check{V}_k(x_j) + (1 - M_{jk}) \hat{V}_k(x_j) \right), \quad (11)$$

where M_{jk} is 1 if node j belongs to particle k and 0 otherwise. As illustrated in Fig. 22, apart from a few saddle points, the node is pulled towards the nodes belonging to the same particle and away from nodes belonging to other particles.

The performance of this algorithm is compared with a baseline PF algorithm in a sparse, low-pileup LHC environment. The proposed method selects more real particles and misidentifies less fake particles than the standard approach.

Similarly, in [43], an end-to-end trainable machine-learned PF (MLPF) algorithm for reconstructing particle candidates is proposed

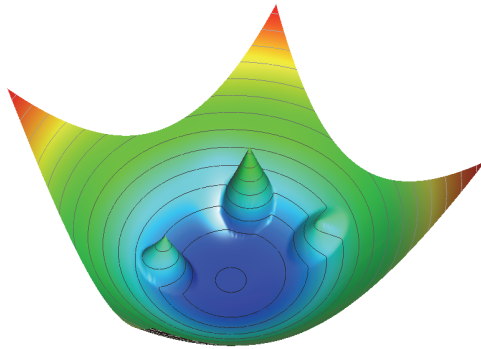


Fig. 22. Illustration of the object condensation loss function combining four effective potentials: three that repel a given node and one in the center that attracts the node [42].

based on a parallelizable, computationally efficient, scalable GNN and a multi-task objective. Given set of detector inputs X , the goal of the algorithm is to predict a set of particle candidates Y' that closely approximates the target generator particle set Y , i.e. minimizing some differentiable set-to-set metric $\|Y - Y'\| \in \mathbb{R}$. The target and predicted sets may have a different number of elements. To simplify the problem numerically, the target set Y can be zero-padded so that $|Y| = |X|$. Then, the loss function can be computed element-by-element:

$$\|Y - Y'\| \equiv \sum_{i \in \text{event}} L(y_i, y'_i), \quad (12)$$

$$L(y_i, y'_i) \equiv \text{CLS}(c_i, c'_i) + \alpha \text{REG}(p_i, p'_i), \quad (13)$$

where the target values and predictions $y_i = [c_i; p_i]$ are decomposed such that the multi-classification (CLS) is encapsulated in the scores and one-hot encoded classes c_i , while the momentum and charge regression (REG) values in p_i . Since the target particles are often geometrically and energetically close to identifiable detector inputs, the target set Y can be arranged such that if a target particle y_i is best associated to a detector input x_i , it is arranged to be in the same location in the sequence. This data preprocessing step is conceptually similar to the object condensation approach [42].

To create the dynamic graph (see Fig. 23) between input detector elements, an approach based on kNN and locality sensitive hashing (LSH) [167] is used to improve the time complexity of the graph building algorithm. The method divides the input set into bins of nearby elements using a hash function, such that constructing a kNN graph in each bin is fast. With the graph built dynamically, message passing is performed along the graph structure to create hidden states of the input elements. Based on a benchmark particle-level dataset generated using PYTHIA8 and DELPHES3, the MLPF GNN reconstruction yields comparable or better physics performance for charged and neutral hadrons relative to the baseline rule-based PF algorithm in DELPHES. The inference time empirically scales

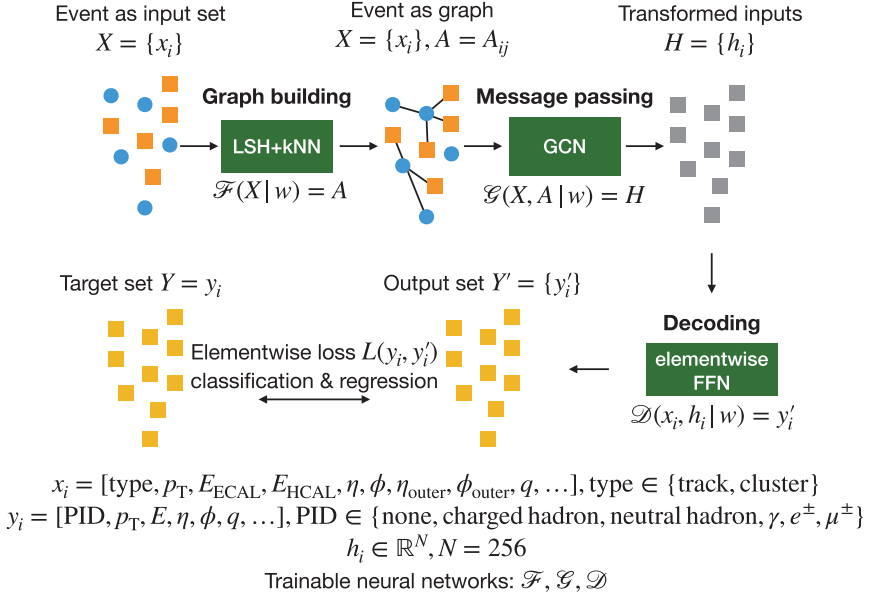


Fig. 23. Schematic of the end-to-end trainable MLPF [43] setup with GNN. The event is represented as a set of detector elements x_i . The set is transformed into a graph by the graph building step, which is implemented using an LSH approximation of kNN. The graph nodes are then encoded using a message passing step, implemented using graph convolutional nets. The encoded elements are decoded to the output feature vectors y_i using pointwise feedforward networks.

approximately linearly with the input size, which is promising for efficient evaluation at the HL-LHC.

6. Summary

Graph neural networks (GNNs) that operate on point clouds and graphs are increasingly popular for applications in high-energy physics (HEP) event reconstruction. One reason for their popularity is a closer correspondence to the input HEP data or the desired output. Namely, measurements in a detector naturally form a point cloud, which can be interpreted as the nodes in a graph once the connectivity (edges) is specified. The solution to many HEP

reconstruction tasks can be mapped onto the edges of the graph (e.g. track finding), the nodes of the graph (e.g. pileup mitigation), or graph characteristics (e.g. jet tagging). Another reason is practical: the computational performance of many traditional reconstruction approaches scales poorly as the collision events become more complex, while GNNs have the potential to scale up better, especially by leveraging highly parallel architectures like graphics processing units or field-programmable gate arrays.

A variety of GNN models have been used for node-level, edge-level, and graph-pooled tasks, and all models share common structures that involve propagating and aggregating information between different nodes in the graph. Another key ingredient is in the construction of the initial graph connectivity and whether that connectivity is dynamic (learned) or static. The physics performance of GNNs has been shown to match or surpass that of state-of-the-art techniques in several proof-of-concept studies. However, many of the models have not yet been tested with real detector data, or benchmarked in terms of their computational performance. Nonetheless, the approach is increasingly promising, as more and more HEP applications continue to appear. At their core, GNNs model the nature of the interactions between the objects in an input set, which may explain why particle physicists, trying to model the nature of the interactions between elementary particles, find them so applicable.

Acknowledgments

We thank Jonathan Shlomi and Peter Battaglia for discussions and sharing materials reproduced here. We thank authors of other chapters for feedback on this one. J. D. is supported by the U.S. Department of Energy (DOE), Office of Science, Office of High Energy Physics Early Career Research program under Award No. DE-SC0021187. J.-R. V. is partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 772369) and by the U.S. DOE, Office of Science, Office of High Energy Physics under Award No. DE-SC0011925, DE-SC0019227, and DE-AC02-07CH11359.

References

- [1] B. H. Denby, Neural networks and cellular automata in experimental high-energy physics, *Comput. Phys. Commun.* **49** (1988) 429.
- [2] A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, A. Himmel, A. Aurisano, K. Terao and T. Wongjirad, Machine learning at the energy and intensity frontiers of particle physics, *Nature* **560** (2018) 41.
- [3] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto and L. Zdeborová, Machine learning and the physical sciences, *Rev. Mod. Phys.* **91** (2019) 045002; arXiv:1903.10563 [physics.comp-ph].
- [4] D. Guest, K. Cranmer and D. Whiteson, Deep learning and its application to LHC physics, *Ann. Rev. Nucl. Part. Sci.* **68** (2018) 161; arXiv:1806.11484 [hep-ex].
- [5] D. Bourilkov, Machine and deep learning applications in particle physics, *Int. J. Mod. Phys. A* **34** (2020) 1930019; arXiv:1912.08245 [physics.data-an].
- [6] A. J. Larkoski, I. Moutl, and B. Nachman, Jet substructure at the Large Hadron Collider: A review of recent advances in theory and machine learning, *Phys. Rept.* **841** (2020) 1; arXiv:1709.04464 [hep-ph].
- [7] HEP Community, A living review of machine learning for particle physics (2020); <https://iml-wg.github.io/HEPML-LivingReview/>.
- [8] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, The graph neural network model, *IEEE Trans. Neural Netw.* **20** (2009) 61.
- [9] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu, Interaction networks for learning about objects, relations and physics, in *Advances in Neural Information Processing Systems 29*, eds. D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Curran Associates, Inc., 2016); arXiv:1612.00222 [cs.AI].
- [10] C. R. Qi, H. Su, K. Mo and L. J. Guibas, PointNet: Deep learning on point sets for 3D classification and segmentation, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017); arXiv:1612.00593 [cs.CV].
- [11] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam and P. Vandergheynst, Geometric deep learning: going beyond Euclidean data, *IEEE Signal Process. Mag.* **34** (2017) 18; arXiv:1611.08097 [cs.CV].
- [12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, Neural message passing for quantum chemistry, in *Proc. 34th Int. Conf. Machine Learning* (2017); arXiv:1704.01212 [cs.LG].
- [13] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein and J. M. Solomon, Dynamic graph CNN for learning on point clouds, *ACM Trans. Graph.* **38** (2019); arXiv:1801.07829.
- [14] P. W. Battaglia *et al.*, Relational inductive biases, deep learning, and graph networks (2018); arXiv:1806.01261 [cs.LG].
- [15] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li and M. Sun, Graph neural networks: A review of methods and applications (2018); arXiv:1812.08434 [cs.LG].

- [16] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and P. S. Yu, A comprehensive survey on graph neural networks, *IEEE Trans. Neural Netw. Learn. Syst.* (2020) 1; arXiv:1901.00596 [cs.LG].
- [17] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li and M. Sun, Graph neural networks: A review of methods and applications (2018); arXiv:1812.08434 [cs.LG].
- [18] Z. Zhang, P. Cui and W. Zhu, Deep learning on graphs: A survey (2018); arXiv:1812.04202 [cs.LG].
- [19] D. Bacciu, F. Errica, A. Micheli and M. Podda, A gentle introduction to deep learning for graphs, *Neural Netw.* **129** (2020) 203.
- [20] S. Farrell *et al.*, Novel deep learning methods for track reconstruction, in *4th Int. Workshop Connecting the Dots 2018* (2018); arXiv:1810.06111 [hep-ex].
- [21] X. Ju *et al.*, Graph neural networks for particle reconstruction in high energy physics detectors, in *Machine Learning and the Physical Sciences Workshop at the 33rd Annual Conf. Neural Information Processing Systems* (2019); arXiv:2003.11603 [physics.ins-det].
- [22] N. Choma *et al.*, Track seeding and labelling with embedded-space graph neural networks, in *6th Int. Workshop Connecting the Dots 2020* (2020); arXiv:2007.00149 [physics.ins-det].
- [23] S. Amrouche, M. Kiehn, T. Golling and A. Salzburger, Tracking aware metric learning for particle reconstruction, in *3rd Machine Learning and the Physical Sciences Workshop at the 34th Annual Conf. Neural Information Processing Systems.* (2020).
- [24] S. Amrouche, M. Kiehn, T. Golling and A. Salzburger, Hashing and metric learning for charged particle tracking, in *2nd Machine Learning and the Physical Sciences Workshop at the 33rd Annual Conf. on Neural Information Processing Systems.* (2019); arXiv:2101.06428 [hep-ex].
- [25] S. Amrouche, T. Golling, M. Kiehn, C. Plant and A. Salzburger, Similarity hashing for charged particle tracking, in *IEEE Int. Conf. on Big Data 2019* (2019).
- [26] I. Henrion, K. Cranmer, J. Bruna, K. Cho, J. Brehmer, G. Louppe and G. Rochette, Neural message passing for jet physics, in *Deep Learning for Physical Sciences Workshop at the 31st Conf. Neural Information Processing Systems.* (Long Beach, CA, 2017).
- [27] P. T. Komiske, E. M. Metodiev and J. Thaler, Energy flow networks: Deep sets for particle jets, *J. High Energy Phys.* **01** (2019) 121; arXiv:1810.05165 [hep-ph].
- [28] E. A. Moreno *et al.*, Interaction networks for the identification of boosted $H \rightarrow b\bar{b}$ decays, *Phys. Rev. D* **102** (2020) 012010; arXiv:1909.12285 [hep-ex].
- [29] E. A. Moreno *et al.*, JEDI-net: A jet identification algorithm based on interaction networks, *Eur. Phys. J. C* **80** (2020) 58; arXiv:1908.05318 [hep-ex].
- [30] H. Qu and L. Gouskos, ParticleNet: Jet tagging via particle clouds, *Phys. Rev. D* **101** (2020) 056019; arXiv:1902.08570 [hep-ph].

- [31] A. Chakraborty, S. H. Lim, M. M. Nojiri and M. Takeuchi, Neural network-based top tagger with two-point energy correlations and geometry of soft emissions, *J. High Energy Phys.* **20** (2020) 111; arXiv:2003.11787 [hep-ph].
- [32] E. Bernreuther, T. Finke, F. Kahlhoefer, M. Krämer and A. Mück, Casting a graph net to catch dark showers, *SciPost Phys.* **10** (2021) 046; arXiv:2006.08639 [hep-ph].
- [33] M. J. Dolan and A. Ore, Equivariant energy flow networks for jet tagging, *Phys. Rev. D* **103** (2021) 074022; arXiv:2012.00964 [hep-ph].
- [34] J. Guo, J. Li and T. Li, The boosted Higgs jet reconstruction via graph neural network, *Phys. Rev. D* **103** (2021) 116025; arXiv:2010.05464 [hep-ph].
- [35] X. Ju and B. Nachman, Supervised jet clustering with graph neural networks for Lorentz boosted bosons, *Phys. Rev. D* **102** (2020) 075014; arXiv:2008.06064 [hep-ph].
- [36] M. Abdughani, J. Ren, L. Wu and J. M. Yang, Probing stop pair production at the LHC with graph neural networks, *J. High Energy Phys.* **08** (2019) 055; arXiv:1807.09088 [hep-ph].
- [37] J. Ren, L. Wu, and J. M. Yang, Unveiling CP property of top-Higgs coupling with graph neural networks at the LHC, *Phys. Lett. B* **802** (2020) 135198; arXiv:1901.05627 [hep-ph].
- [38] M. Abdughani, D. Wang, L. Wu, J. M. Yang and J. Zhao, Probing triple Higgs coupling with machine learning at the LHC (2020); arXiv:2005.11086 [hep-ph].
- [39] S. R. Qasim, J. Kieseler, Y. Iiyama and M. Pierini, Learning representations of irregular particle-detector geometry with distance-weighted graph networks, *Eur. Phys. J. C* **79** (2019) 608; arXiv:1902.07987 [physics.data-an].
- [40] H. Serviansky, N. Segol, J. Shlomi, K. Cranmer, E. Gross, H. Maron and Y. Lipman, Set2Graph: Learning graphs from sets, in *Graph Representation Learning and Beyond Workshop at the 37th Int. Conf. Machine Learning.* (2020); arXiv:2002.08772 [cs.LG].
- [41] J. Shlomi, S. Ganguly, E. Gross, K. Cranmer, Y. Lipman, H. Serviansky, H. Maron and N. Segol, Secondary vertex finding in jets with neural networks, *Eur. Phys. J. C* **81** (2021) 540; arXiv:2008.02831 [hep-ex].
- [42] J. Kieseler, Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph and image data, *Eur. Phys. J. C* **80** (2020) 886; arXiv:2002.03605 [physics.data-an].
- [43] J. Pata, J. Duarte, J.-R. Vlimant, M. Pierini and M. Spiropulu, MLPF: Efficient machine-learned particle-flow reconstruction using graph neural networks, *Eur. Phys. J. C* **81** (2021) 381; arXiv:2101.08578 [physics.data-an].
- [44] J. Arjona Martínez, O. Cerri, M. Pierini, M. Spiropulu and J.-R. Vlimant, Pileup mitigation at the large hadron collider with graph neural networks, *Eur. Phys. J. Plus* **134** (2019) 333; arXiv:1810.07988 [hep-ph].
- [45] V. Mikuni and F. Canelli, ABCNet: An attention-based method for particle tagging, *Eur. Phys. J. Plus* **135** (2020) 463; arXiv:2001.05311 [physics.data-an].

- [46] J. Shlomi, P. Battaglia and J.-R. Vlimant, Graph neural networks in particle physics, *Mach. Learn.: Sci. Tech.* (2020) 021001; arXiv:2007.13681 [hep-ex].
- [47] R. H. Dennard, F. H. Gaensslen, H. Yu, V. L. Rideout, E. Bassous and A. R. LeBlanc, Design of ion-implanted MOSFET's with very small physical dimensions, *IEEE J. Solid-State Circuits* **9** (1974) 256.
- [48] H. Esmailzadeh, E. Blem, R. St. Amant, K. Sankaralingam and D. Burger, Dark silicon and the end of multicore scaling, in *Proc. of the 38th Annual Int. Symp. Computer Architecture* (ACM, New York, NY, 2011).
- [49] C. Jones, P. Elmer, L. Sexton-Kennedy, C. Green and A. Baldoocci, Multi-core aware applications in CMS, *J. Phys. Conf. Ser.* **331** (2011) 042012.
- [50] J. Hernandez, D. Evans and S. Foulkes, Multi-core processing and scheduling performance in CMS, *J. Phys. Conf. Ser.* **396** (2012) 032055.
- [51] CMS, C. Jones, L. Contreras, P. Gartung, D. Hufnagel and L. Sexton-Kennedy, Using the CMS threaded framework in a production environment, *J. Phys. Conf. Ser.* **664** (2015) 072026.
- [52] D. Riley and C. Jones, Multi-threaded output in CMS using ROOT, *Eur. Phys. J. Web Conf.* **214** (2019) 02016; arXiv:1905.02113 [cs.DC].
- [53] A. Bocci, D. Dagenhart, V. Innocente, C. Jones, M. Kortelainen, F. Pantaleo and M. Rovere, Bringing heterogeneity to the CMS software framework, *Eur. Phys. J. Web Conf.* **245** (2020) 05009; arXiv:2004.04334 [physics.comp-ph].
- [54] HEP Software Foundation, J. Albrecht *et al.*, A roadmap for HEP software and computing R&D for the 2020s, *Comput. Softw. Big Sci.* **3** (2019) 7; arXiv:1712.06982 [physics.comp-ph].
- [55] CMS, CMS offline and computing public results, (2020); <https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults>.
- [56] J. Cogan, M. Kagan, E. Strauss and A. Schwartzman, Jet-images: Computer vision inspired techniques for jet tagging, *J. High Energy Phys.* **02** (2015) 118; arXiv:1407.5675 [hep-ph].
- [57] ATLAS, Quark versus gluon jet tagging using jet images with the ATLAS detector, ATLAS Note, ATL-PHYS-PUB-2017-017 (2017).
- [58] G. Kasieczka, T. Plehn, M. Russell and T. Schell, Deep-learning top taggers or the end of QCD?, *J. High Energy Phys.* **05** (2017) 006; arXiv:1701.08784 [hep-ph].
- [59] S. Macaluso and D. Shih, Pulling out all the tops with computer vision and deep learning, *J. High Energy Phys.* **10** (2018) 121; arXiv:1803.00107 [hep-ph].
- [60] M. Andrews, M. Paulini, S. Gleyzer and B. Poczos, End-to-end physics event classification with CMS open data: Applying image-based deep learning to detector data for the direct classification of collision events at the LHC, *Comput. Softw. Big Sci.* **4** (2020) 6; arXiv:1807.11916 [hep-ex].
- [61] J. Lin, M. Freytsis, I. Moulton and B. Nachman, Boosting $H \rightarrow b\bar{b}$ with machine learning, *J. High Energy Phys.* **10** (2018) 101; arXiv:1807.10768 [hep-ph].

- [62] ATLAS, Convolutional neural networks with event images for pileup mitigation with the ATLAS detector, ATLAS Note, ATL-PHYS-PUB-2019-028 (2019).
- [63] ATLAS, Identification of jets containing b -hadrons with recurrent neural networks at the ATLAS experiment, ATLAS Note, ATL-PHYS-PUB-2017-003 (2017).
- [64] CMS, A. M. Sirunyan *et al.*, Identification of heavy, energetic, hadronically decaying particles using machine-learning techniques, *J. Instrum.* **15** (2020) P06005; arXiv:2004.08262 [hep-ex].
- [65] G. Louppe, K. Cho, C. Becot and K. Cranmer, QCD-aware recursive neural networks for jet physics, *J. High Energy Phys.* **01** (2019) 057; arXiv:1702.00748 [hep-ph].
- [66] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman and A. Schwartzman, Jet-images — deep learning edition, *J. High Energy Phys.* **07** (2016) 069; arXiv:1511.05190 [hep-ph].
- [67] J. Pearkes, W. Fedorko, A. Lister and C. Gay, Jet constituents for deep neural network based top quark tagging (2017); arXiv:1704.02124 [hep-ex].
- [68] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov and A. J. Smola, Deep sets, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017); arXiv:1703.06114 [cs.LG].
- [69] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. Battaglia and T. Lillicrap, A simple neural network module for relational reasoning, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (Curran Associates, Inc., 2017); arXiv:1706.01427 [cs.CL].
- [70] X. Wang, R. Girshick, A. Gupta and K. He, Non-local neural networks, in *2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition* (2018); arXiv:1711.07971 [cs.CV].
- [71] Y. Li, O. Vinyals, C. Dyer, R. Pascanu and P. Battaglia, Learning deep generative models of graphs, in *6th Int. Conf. Learning Representations, Workshop Track* (2018); arXiv:1803.03324 [cs.LG].
- [72] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling and R. Zemel, Neural relational inference for interacting systems, in *Proc. 35th International Conf. Machine Learning*, eds. J. Dy and A. Krause (PMLR, Stockholmsmässan, Stockholm Sweden, 2018); arXiv:1802.04687 [stat.ML].
- [73] T. M. Mitchell, The need for biases in learning generalizations, *Technical Report*, Rutgers University, New Brunswick, NJ (1980); http://www-cgi.cs.cmu.edu/~tom/pubs/NeedForBias_1980.pdf.
- [74] E. Wagstaff, F. B. Fuchs, M. Engelcke, I. Posner and M. Osborne, On the limitations of representing functions on sets, in *Proc. 36th International Conf. Machine Learning*, eds. K. Chaudhuri and R. Salakhutdinov (PMLR, Long Beach, CA, 2019); arXiv:1901.09006 [cs.LG].

- [75] R. Kondor and S. Trivedi, On the generalization of equivariance and convolution in neural networks to the action of compact groups, in *Proc. 35th Int. Conf. Machine Learning*, eds. J. Dy and A. Krause, PMLR (Stockholmsmässan, Stockholm Sweden, 2018); arXiv:1802.03690 [stat.ML].
- [76] H. Maron, H. Ben-Hamu, N. Shamir and Y. Lipman, Invariant and equivariant graph networks, in *Int. Conf. Learning Representations* (2019); arXiv:1812.09902 [cs.LG].
- [77] N. Keriven and G. Peyré, Universal invariant and equivariant graph neural networks, in *Advances in Neural Information Processing Systems 32*, eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox and R. Garnett (2019); arXiv:1905.04943 [cs.LG].
- [78] A. Sannai, Y. Takai and M. Cordonnier, Universal approximations of permutation invariant/equivariant functions by deep neural networks (2019); arXiv:1903.01939 [cs.LG].
- [79] A. Bogatskiy, B. Anderson, J. T. Offermann, M. Roussi, D. W. Miller and R. Kondor, Lorentz group equivariant neural network for particle physics, in *Proc. 37th Int. Conf. Machine Learning*, eds. H. Daumé and A. Singh (2020); arXiv:2006.04780 [hep-ph].
- [80] B. K. Miller, M. Geiger, T. E. Smidt and F. Noé, Relevance of rotationally equivariant convolutions for predicting molecular properties, in *NeurIPS Workshop on Interpretable Inductive Biases and Physically Structured Learning* (2020); arXiv:2008.08461 [cs.LG].
- [81] T. Smidt, Euclidean symmetry and equivariance in machine learning, *Trends Chem* **3**(2) (2021) 82–85; ChemRxiv:12935198;
- [82] N. Dym and H. Maron, On the universality of rotation equivariant point cloud networks, in *9th Int. Conf. Learning Representations* (2021); arXiv:2010.02449 [cs.LG].
- [83] G. Yehudai, E. Fetaya, E. Meirom, G. Chechik and H. Maron, On size generalization in graph neural networks, in *Int. Conf. Learning Representations* (2021); arXiv:2010.08853 [cs.LG].
- [84] B. Knyazev, G. W. Taylor and M. Amer, Understanding attention and generalization in graph neural networks, in *Advances in Neural Information Processing Systems 32*, eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox and R. Garnett (Curran Associates, Inc., 2019); arXiv:1905.02850 [cs.LG].
- [85] S. Verma and Z.-L. Zhang, Stability and generalization of graph convolutional neural networks, in *Proc. 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (ACM, New York, NY, 2019). arXiv:1905.01004 [cs.LG].
- [86] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, Attention is all you need, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (Curran Associates, Inc., 2017); arXiv:1706.03762.

- [87] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò and Y. Bengio, Graph attention networks, in *6th Int. Conf. Learning Representations* (2018). arXiv:1710.10903 [stat.ML].
- [88] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton and J. Leskovec, Hierarchical graph representation learning with differentiable pooling, in *Advances in Neural Information Processing Systems 31*, eds. S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett (Curran Associates, Inc., 2018); arXiv:1806.08804 [cs.LG].
- [89] H. Gao and S. Ji, Graph U-nets, in *Proc. 36th Int. Conf. Machine Learning*, eds. K. Chaudhuri and R. Salakhutdinov (PMLR, Long Beach, CA, 2019); arXiv:1905.05178 [cs.LG].
- [90] J. Lee, I. Lee and J. Kang, Self-attention graph pooling, in *Proc. 36th Int. Conf. Machine Learning*, eds. K. Chaudhuri and R. Salakhutdinov (PMLR, Long Beach, CA, 2019); arXiv:1904.08082 [cs.LG].
- [91] F. Diehl, T. Brunner, M. T. Le and A. Knoll, Towards graph pooling by edge contraction, in *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data* (2019); <https://graphreason.github.io/papers/17.pdf>.
- [92] I. S. Dhillon, Y. Guan and B. Kulis, Weighted graph cuts without eigenvectors a multilevel approach, *IEEE Trans. Pattern Anal. Mach. Intell.* **29** (2007) 1944.
- [93] D. Bacciu and L. D. Sotito, A non-negative factorization approach to node pooling in graph convolutional neural networks, in *AI*IA 2019 — Advances in Artificial Intelligence*, M. Alviano, G. Greco and F. Scarcello (Springer, Cham, 2019); arXiv:1909.03287 [cs.LG].
- [94] L. Gray, T. Klijsma and S. Ghosh, A dynamic reduction network for point clouds (2020); arXiv:2003.08013 [cs.CV].
- [95] B. O. Fagginger Auer and R. H. Bisseling, A GPU algorithm for greedy graph matching, in *Facing the Multicore-Challenge II: Aspects of New Paradigms and Technologies in Parallel Computing* (Springer, Berlin, 2012), p.108.
- [96] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda and M. M. Bronstein, Geometric deep learning on graphs and manifolds using mixture model CNNs (2016); arXiv:1611.08402 [cs.CV].
- [97] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles and H. C. Wolf, Parametric correspondence and Chamfer matching: Two new techniques for image matching, in *Proc. 5th Int. Joint Conference on Artificial Intelligence (IJCAI)* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1977).
- [98] H. Fan, H. Su and L. J. Guibas, A point set generation network for 3D object reconstruction from a single image in *2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (2017); arXiv:1612.00603 [cs.CV].
- [99] Y. Zhang, J. Hare and A. Prügel-Bennett, FSPool: Learning set representations with featurewise sort pooling, in *8th Int. Conf. Learning Representations* (2020); arXiv:1906.02795 [cs.LG].

- [100] Y. Zhang, J. Hare and A. Prügel-Bennett, Deep set prediction networks, in *Advances in Neural Information Processing Systems 32*, eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett (Curran Associates, Inc., 2019); arXiv:1906.06565 [cs.LG].
- [101] S. Peleg, M. Werman and H. Rom, A unified approach to the change of resolution: Space and gray-level, *IEEE Trans. Pattern Anal. Mach. Intell.* **11** (1989) 739.
- [102] P. T. Komiske, E. M. Metodiev and J. Thaler, Metric space of collider events, *Phys. Rev. Lett.* **123** (2019) 041801; arXiv:1902.02346 [hep-ph].
- [103] Y. LeCun, J. S. Denker and S. A. Solla, Optimal brain damage, in *Advances in Neural Information Processing Systems 2*, ed. D. S. Touretzky (Morgan-Kaufmann, 1990).
- [104] S. Han, H. Mao and W. J. Dally, Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding, in *4th Int. Conf. Learning Representations, ICLR 2016, Conference Track Proc.* (2016); arXiv:1510.00149 [cs.CV].
- [105] T. Yang, Y. Chen and V. Sze, Designing energy-efficient convolutional neural networks using energy-aware pruning, in *2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (2017); arXiv:1611.05128 [cs.CV].
- [106] S. Han, J. Pool, J. Tran and W. J. Dally, Learning both weights and connections for efficient neural networks, in *Advances in Neural Information Processing Systems 28*, eds. C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett (Curran Associates, Inc., 2015); arXiv:1506.02626 [cs.NE].
- [107] C. Louizos, M. Welling and D. P. Kingma, Learning sparse neural networks through L_0 regularization, in *6th Int. Conf. Learning Representations* (2018); arXiv:1712.01312 [stat.ML].
- [108] J. Frankle and M. Carbin, The lottery ticket hypothesis: Training pruned neural networks, in *7th Int. Conf. Learning Representation* (2019); arXiv:1803.03635.
- [109] J. Duarte *et al.*, Fast inference of deep neural networks in FPGAs for particle physics, *J. Instrum.* **13** (2018) P07027; arXiv:1804.06913 [physics.ins-det].
- [110] G. Di Guglielmo *et al.*, Compressing deep neural networks on FPGAs to binary and ternary precision with `hls4ml`, *Mach. Learn.: Sci. Technol.* **2** (2020) 015001.
- [111] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, T. Aarrestad, V. Loncar, J. Ngadiuba, M. Pierini, A. A. Pol and S. Summers, Automatic deep heterogeneous quantization of deep neural networks for ultra low-area, low-latency inference on the edge at particle colliders, *Nat. Mach. Intell.* (2021); doi: 10.1038/s42256-021-00356-5; arXiv:2006.10159 [physics.ins-det].
- [112] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally and K. Keutzer, SqueezeNet: AlexNet-level accuracy with $50\times$ fewer parameters and < 1 MB model size, preprint (2016); arXiv:1602.07360 [cs.CV].

- [113] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, MobileNets: Efficient convolutional neural networks for mobile vision applications (2017); arXiv:1704.04861 [cs.CV].
- [114] Z. Liu, H. Tang, Y. Lin and S. Han, Point-voxel CNN for efficient 3D deep learning, in *Advances in Neural Information Processing Systems 32*, eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox and R. Garnett (Curran Associates, Inc., 2019).
- [115] M. Fey and J. E. Lenssen, Fast graph representation learning with PyTorch Geometric, in *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019); arXiv:1903.02428 [cs.LG]; <https://pytorch-geometric.readthedocs.io/>.
- [116] M. Wang *et al.*, Deep Graph Library: Towards efficient and scalable deep learning on graphs, in *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019); arXiv:1909.01315 [cs.LG]; <https://www.dgl.ai>.
- [117] DeepMind, graph_nets (2019); https://github.com/deepmind/graph_nets.
- [118] DeepMind, jraph (2020); <https://github.com/deepmind/jraph>.
- [119] C. Data61, Stellargraph (2018); <https://github.com/stellargraph/stellargraph>.
- [120] D. Grattarola and C. Alippi, Graph neural networks in tensorflow and keras with spektral, in *Graph Representation Learning and Beyond — ICML 2020 Workshop* (2020); arXiv:2006.12138 [cs.LG]; <https://grlplus.github.io/papers/9.pdf>.
- [121] D. Grattarola, Spektral (2020); <https://github.com/danielegrattarola/spektral>.
- [122] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre and K. Vissers, FINN: A framework for fast, scalable binarized neural network inference, in *Proc. 2017 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays* (ACM, New York, NY, 2017); arXiv:1612.07119.
- [123] M. Blott, T. Preußner, N. Fraser, G. Gambardella, K. O'Brien and Y. Umuroglu, FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks, *ACM Trans. Reconfigurable Technol. Syst.* **11** (12, 2018); arXiv:1809.04570 [cs.AR].
- [124] A. Shawahna, S. M. Sait and A. El-Maleh, FPGA-based accelerators of deep learning networks for learning and classification: A review, *IEEE Access* **7** (2019) 7823; arXiv:1901.00121.
- [125] T. Wang, C. Wang, X. Zhou and H. Chen, An overview of FPGA based deep learning accelerators: Challenges and opportunities, in *2019 IEEE 21st Int. Conf. High Performance Computing and Communications; IEEE 17th Int. Conf. Smart City; IEEE 5th Int. Conf. Data Science and Systems (HPCC/SmartCity/DSS)* (2019); arXiv:1901.04988.
- [126] S. Summers *et al.*, Fast inference of boosted decision trees in FPGAs for particle physics, *J. Instrum.* **15** (2020) P05026; arXiv:2002.02534 [physics.comp-ph].

- [127] T. Åarrestad *et al.*, Fast convolutional neural networks on FPGAs with `hls4ml`, *Mach. Learn. Sci. Technol.* (2021); doi:10.1088/2632-2153/ac0ea1; arXiv:2101.05108 [cs.LG].
- [128] Y. Iiyama *et al.*, Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics, *Front. Big Data* **3** (2021) 44; arXiv:2008.03601 [hep-ex].
- [129] A. Heintz, V. Razavimaleki, J. Duarte, G. DeZoort, I. Ojalvo, S. Thais, M. Atkinson, M. Neubauer, L. Gray, S. Jindariani, N. Tran, P. Harris, D. Rankin, T. Aarrestad, V. Loncar, M. Pierini, S. Summers, J. Ngadiuba, M. Liu, E. Kreinar and Z. Wu, Accelerated charged particle tracking with graph neural networks on FPGAs, in *3rd Machine Learning and the Physical Sciences Workshop at the 34th Annual Conf. Neural Information Processing Systems*. (2020); arXiv:2012.01563 [physics.ins-det].
- [130] J. Duarte *et al.*, FPGA-accelerated machine learning inference as a service for particle physics computing, *Comput. Softw. Big Sci.* **3** (2019) 13; arXiv:1904.08986 [physics.data-an].
- [131] J. Krupa *et al.*, GPU coprocessors as a service for deep learning inference in high energy physics, *Mach. Learn. Sci. Technol.* **2** (2021) 035005; doi: 10.1088/2632-2153/abec21; arXiv:2007.10359 [physics.comp-ph].
- [132] D. S. Rankin *et al.*, FPGAs-as-a-service toolkit (FaaSST), in *2020 IEEE/ACM Int. Workshop on Heterogeneous High-Performance Reconfigurable Computing (H2RC)* (2020); arXiv:2010.08556 [physics.comp-ph].
- [133] M. Besta, D. Stanojevic, J. de Fine Licht, T. Ben-Nun and T. Hoeffler, Graph processing on FPGAs: Taxonomy, survey, challenges; arXiv:1903.06697 [cs.DC].
- [134] S. Amrouche *et al.*, The tracking machine learning challenge: Accuracy phase, in *The NeurIPS '18 Competition*. (2020); arXiv:1904.06778 [hep-ex].
- [135] A. Strandlie and R. Frühwirth, Track and vertex reconstruction: From classical to adaptive methods, *Rev. Mod. Phys.* **82** (2010) 1419.
- [136] CMS, S. Chatrchyan *et al.*, Description and performance of track and primary-vertex reconstruction with the CMS tracker, *J. Instrum.* (2014) P10009; arXiv:1405.6569 [physics.ins-det].
- [137] ATLAS, M. Aaboud *et al.*, Performance of the ATLAS track reconstruction algorithms in dense environments in LHC Run 2, *Eur. Phys. J. C* **77** (2017) 673; arXiv:1704.07983 [hep-ex].
- [138] P. Billoir, Progressive track recognition with a Kalman-like fitting procedure, *Comput. Phys. Comm.* **57** (1989) 390.
- [139] P. Billoir and S. Qian, Simultaneous pattern recognition and track fitting by the Kalman filtering method, *Nucl. Instrum. Methods Phys. Res. A* **294** (1990) 219.
- [140] R. Mankel, A concurrent track evolution algorithm for pattern recognition in the HERA-B main tracking system, *Nucl. Instrum. Methods Phys. Res. A* **395** (1997) 169.
- [141] R. Frühwirth, Application of Kalman filtering to track and vertex fitting, *Nucl. Instrum. Methods Phys. Res. A* **262** (1987) 444.

- [142] I. Tomalin *et al.*, An FPGA based track finder for the L1 trigger of the CMS experiment at the high luminosity LHC, *J. Instrum.* **12** (2017) P12019.
- [143] F. Dietrich, Track seed classification with deep neural networks, in *Connecting the Dots and Workshop on Intelligent Trackers*, preprint (2019); arXiv:1910.06779 [physics.ins-det].
- [144] W. Waltenberger, Adaptive vertex reconstruction, CMS Note, CMS-NOTE-2008-033 (2008); <https://cds.cern.ch/record/1166320>.
- [145] W. Waltenberger, R. Frühwirth and P. Vanlaer, Adaptive vertex fitting, *J. Phys. G* **34** (2007) N343.
- [146] W. Waltenberger, RAVE—A detector-independent toolkit to reconstruct vertices, *IEEE Trans. Nucl. Sci.* **58** (2011) 434.
- [147] CMS, A. Sirunyan *et al.*, Identification of heavy-flavour jets with the CMS detector in pp collisions at 13 TeV, *J. Instrum.* **13** (2018) P05011; arXiv:1712.07158 [physics.ins-det].
- [148] ATLAS, S. Heer, The secondary vertex finding algorithm with the ATLAS detector, *Proc. Sci.* **EPS-HEP2017** (2017) 762.
- [149] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, Weisfeiler and Leman go neural: higher-order graph neural networks, in *33rd AAAI Conf. Artificial Intelligence (AAAI 2019)* (2019); arXiv:1810.02244 [cs.LG].
- [150] S. Zagoruyko and N. Komodakis, Learning to compare image patches via convolutional neural networks, in *2015 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2015); arXiv:1504.03641 [cs.CV].
- [151] K. Q. Weinberger, J. Blitzer and L. K. Saul, Distance metric learning for large margin nearest neighbor classification, in *Advances in Neural Information Processing Systems 18*, eds. Y. Weiss, B. Schölkopf and J. C. Platt, (MIT Press, 2006).
- [152] F. Schroff, D. Kalenichenko and J. Philbin, FaceNet: A unified embedding for face recognition and clustering, in *2015 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (2015); arXiv:1503.03832 [cs.CV].
- [153] G. Piacquadio and C. Weiser, A new inclusive secondary vertex algorithm for b-jet tagging in ATLAS, *J. Phys. Conf. Ser.* **119** (2008) 032032.
- [154] L. Hubert and P. Arabie, Comparing partitions, *J. Classif.* **2** 1, 193.
- [155] FCC, A. Abada *et al.*, FCC-hh: The hadron collider, *Eur. Phys. J.* **228** (2019) 755.
- [156] D. Bertolini, P. Harris, M. Low and N. Tran, Pileup per particle identification, *J. High Energy Phys.* **10** (2014) 059; arXiv:1407.6013 [hep-ph].
- [157] Y. Li, D. Tarlow, M. Brockschmidt and R. S. Zemel, Gated graph sequence neural networks, in *4th Int. Conf. Learning Representations, ICLR 2016, Conference Track Proc.*, eds. Y. Bengio and Y. LeCun (2016); arXiv:1511.05493.
- [158] C. Chen, L. Z. Fragonara and A. Tsourdos, GAPNet: Graph attention based point neural network for exploiting local feature of point cloud (2019); arXiv:1905.08705 [cs.CV].
- [159] C. W. Fabjan and T. Ludlam, Calorimetry in high-energy physics, *Annu. Rev. Nucl. Part. Sci.* **32** (1982) 335.

- [160] R. Wigmans, Calorimetry in high energy physics, in *Techniques and Concepts of High-Energy Physics VI*, ed. T. Ferbel, (Springer, Boston, MA, 1991).
- [161] C. Fabjan and F. Gianotti, Calorimetry for particle physics, *Rev. Mod. Phys.* **75** (2003) 1243.
- [162] CMS, S. Chatrchyan *et al.*, The CMS experiment at the CERN LHC, *J. Instrum.* **3** (2008) S08004.
- [163] CMS, A. Sirunyan *et al.*, Particle-flow reconstruction and global event description with the CMS detector, *J. Instrum.* **12** (2017) P10003; arXiv:1706.04965 [physics.ins-det].
- [164] ALEPH, D. Buskulic *et al.*, Performance of the ALEPH detector at LEP, *Nucl. Instrum. Methods Phys. Res. A* **360** (1995) 481.
- [165] ATLAS, M. Aaboud *et al.*, Jet reconstruction and performance using particle flow with the ATLAS Detector, *Eur. Phys. J. C* **77** (2017) 466; arXiv:1703.10485 [hep-ex].
- [166] F. A. Di Bello, S. Ganguly, E. Gross, M. Kado, M. Pitt, J. Shlomi and L. Santi, Towards a computer vision particle flow, *Eur. Phys. J. C* **81**(2) (2021) 107; doi:10.1140/epjc/s10052-021-08897-0; arXiv:2003.08863 [hep-ex].
- [167] N. Kitaev, L. Kaiser and A. Levskaya, Reformer: The efficient transformer, in *8th Int. Conf. Learning Representations* (2020); arXiv:2001.04451 [cs.LG].

Part VI

**Jet Classification and Particle
Identification from Low Level**

This page intentionally left blank

Chapter 13

Image-Based Jet Analysis

Michael Kagan

*SLAC National Accelerator Laboratory,
2575 Sandhill Rd, Menlo Park, CA, 94025, USA
makagan@slac.stanford.edu*

Image-based jet analysis is built upon the *jet image* representation of jets that enables a direct connection between high-energy physics and the fields of computer vision and deep learning. Through this connection, a wide array of new jet analysis techniques have emerged. In this text, we survey jet image-based classification models, built primarily on the use of convolutional neural networks, examine the methods to understand what these models have learned and what is their sensitivity to uncertainties, and review the recent successes in moving these models from phenomenological studies to real-world application on experiments at the LHC. Beyond jet classification, several other applications of jet image-based techniques, including energy estimation, pileup noise reduction, data generation, and anomaly detection, are discussed.

1. Introduction

The *jet image* [1] approach to jet tagging is built upon the rapidly developing field of computer vision (CV) in machine learning (ML). Jets [2, 3] are collimated streams of particles produced by the fragmentation and hadronization of high-energy quarks and gluons. The particles are subsequently measured by particle detectors and clustered with jet clustering algorithms to define the jets. Jet images view the energy depositions of the stream of particles comprising a jet within a fixed geometric region of a detector as an image, thereby connecting particle detector measurements with an image representation and allowing the application of image analysis techniques

from CV. In this way, models built upon advancements in deep convolutional neural networks (CNN) can be trained for jet classification, energy determination through regression, and the reduction of noise, e.g. from simultaneous background interactions at a high intensity hadron collider such as the Large Hadron Collider (LHC). Throughout this text, the focus will be on the use of jet image techniques studied within the context of hadron colliders like the LHC [4].

Jet images form a representation of jets highly connected with the detector; one can look at segmented detectors as imaging devices and interpret the measurements as an image. In contrast, other representations of jets exist that are built more closely from the physics of jet formation, such as viewing jets as sequences [5, 6] or trees [7] formed through a sequential emission process, or viewing jets as sets, graphs, or point clouds [8, 9] with the geometric relationship between constituents of the jet encoded in the adjacency matrix and node properties. There are overlaps in these approaches, for instance a graph can be defined over detector energy measurements, but these approaches will not be discussed in detail in this chapter. The utilization of an image-based approach comes with the major advantage that CV is a highly developed field of ML with some of the most advanced models available for application to jet analysis with jet images. From the experimental viewpoint, the detector measurements are fundamental to any subsequent analysis, and the detailed knowledge of the detector and its systematic uncertainties can be highly advantageous for analysis of LHC data.

Among the earliest use of jet images was for the classification of the parent particle inducing the jet [1], and relied on utilizing linear discriminants trained on image representations of jets for this task. While the remainder of this text will focus on deep learning approaches to jet images, even this early work saw interesting discrimination power for this task. By utilizing the detector measurements directly, rather than relying on jet features developed using physics domain knowledge, additional discrimination power could be extracted. Deep learning approaches surpass such linear methods, but build on this notion of learning discriminating information from detector observables rather than engineered features.

While designed to take advantage of advances in computer vision, jet images have notable differences with respect to typical natural images in CV. Jet images are sparse, with most pixels in the image having zero content. This is markedly different from natural images that tend to have all pixels containing content. Moreover, jet images tend to have multiple localized regions of high density in addition to diffusely located pixels throughout the image, as opposed to the smooth structures typically found in natural images. An example top quark jet image illustrating these features can be seen in Fig. 1. These differences can lead to notable challenges, for instance the number of parameters used in jet image models (and consequently the training time) tends to be large to account for the size of the image, even though most pixels carry no information. Some techniques exist for sparse-image computer vision approaches [11], but have not been explored in depth within the jet image community.

This text will first discuss jets and typical jet physics in Sec. 2. The formation of jet images and the jet image preprocessing steps before

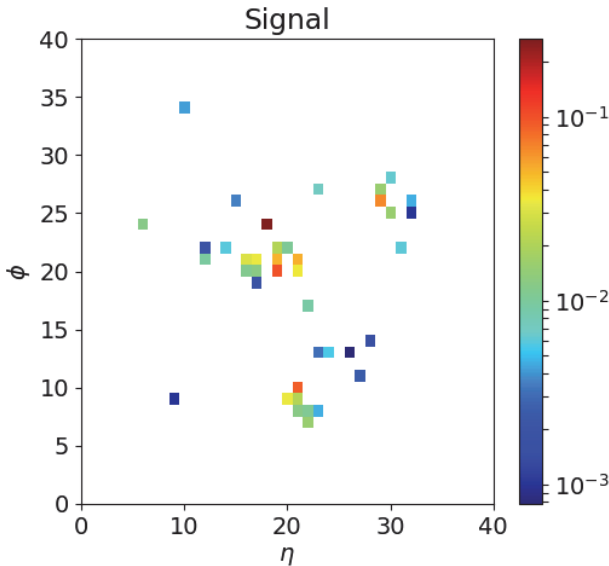


Fig. 1. An example jet image of a Lorentz boosted top quark jet after preprocessing has been applied [10].

classification are discussed in Sec. 3. A brief introduction to computer vision is found in Sec. 4. The application of jet images in various jet classification problems is then discussed in Sec. 5, followed by a discussion on the interpretation of information learned by jet image-based classifiers in Sec. 6. Some recent applications of jet images beyond classification are discussed in Sec. 7. A brief note on the notation used throughout the text follows below (Sec. 1.1).

It should be noted that the majority of the studies presented in this text relate to phenomenological work using simplified setting that often do not include a realistic modeling of a detector's impact on observables. These will frequently be denoted as *phenomenological studies*, in contrast to the studies using realistic detector simulations or using real experiment data that are discussed mainly in Sec. 5.4.

1.1. Notations and definitions

As we will focus on studies of jet images in the LHC setting, we will primarily utilize the hadron collider coordinate system notation. The beam-line defines the z -axis, ϕ indicates the azimuthal angle, $\eta = -\log \tan \frac{\theta}{2}$ is the pseudo-rapidity which is a transformation of the polar angle θ . The rapidity is defined as $y = \frac{1}{2} \log \left[\frac{E+p_z}{E-p_z} \right]$ is frequently used for the polar measurement of massive particles, such as jets, as differences in rapidity are invariant with respect to Lorentz boosts along the beam direction. The angular separation of particles is defined as $\Delta R(p_1, p_2) = \sqrt{(y_1 - y_2)^2 + (\phi_1 - \phi_2)^2}$. The transverse momentum $p_T = \sqrt{p_x^2 + p_y^2}$ is frequently used as it is invariant with respect to Lorentz boosts along the beam direction. The transverse energy is defined as $E_T = E \sin(\theta)$.

2. Jets and Jet Physics Challenges

Jets are collimated streams of particles produced by the fragmentation and hadronization of high-energy quarks and gluons. Jet clustering algorithms are used to combine particles into clusters that define the jets (see [2, 3] for recent reviews). At the LHC, jet algorithms typically rely on sequential reclustering algorithms which,

given a definition of distance, iteratively combine the closest two constituents (either particles or previously combined sets of particles denoted *proto-jets*) until a stopping condition is met. Different distance metrics define different jet algorithms and perhaps the most commonly used algorithm at the LHC is the anti- k_T algorithm [12] in which the distance between particle i and particle j is defined as $d_{ij} = \min\{k_{T,i}^{-2}, k_{T,j}^{-2}\} \Delta_{ij}^2 / R^2$. Here, $\Delta_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$ and y , ϕ , and k_T are the particle rapidity, azimuth, and transverse momentum, respectively. The parameter R of the jet algorithm has the effect of defining the spatial span, or approximate “radius” (though the jet is not necessarily circular), of the jet. Jets and jet algorithms are required to be *IRC safe*, i.e. insensitive to additional infrared radiation or collinear splittings of particles, in order for the jet properties to be calculable in fixed-order perturbation theory. This allows comparison between jets clustered on partons from the hard scattering process, referred to as parton jets, on final state particles after showering and hadronization simulation, referred to as particle jets, and on reconstructed particles in detectors, referred to as reconstructed jets.

Most of the work presented in this text are phenomenological studies outside the context of any individual experiment. These studies primarily utilize particle level simulation after fragmentation and hadronization and thus study particle jets defined after clustering the final state particles. These studies typically do not use a simulation of a detector and its impact on particle kinematic measurements. Studies of jets and jet images after real detector simulation or in real detector data are discussed in Sec. 5.4. In the detector setting, various inputs to jet algorithms can be used to define jets: (1) towers refer to a fixed spatial extent in η and ϕ in which all energy within the longitudinal depth of the calorimeter is summed, (2) topological clusters [13] are used to cluster together energy depositions in nearby calorimeter cells, (3) tracks, or charged particle trajectories, measured using tracking detectors. The particle flow (PF) algorithm [14] is used by the CMS collaboration to match charged particles with energy in the calorimeter in order to utilize both measurements to define PF candidates that can be used as inputs to jet algorithms.

The R -parameter of the jet is used to define the spatial extent to which particles are clustered into the jet. When studying quark and gluon jets, $R = 0.4$ is frequently used. When studying the decay of Lorentz boosted heavy particles, in which multiple partons may be spatially collimated, *large- R* jets are often used which have a larger $R = 1.0$ or $R = 1.2$. *Subjects*, defined by running a jet clustering algorithm with smaller radius on the constituents of a jet, are frequently used to study the internal properties of a jet. More broadly, *jet substructure* refers to the study of the internal structure of jets and the development of theoretically motivated jet features which are useful for discrimination and inference tasks (see [15, 16] for recent reviews).

One particularly important feature of a jet is the jet mass, computed as: $m^2 = (\sum_{i \in \text{jet}} p_i)^2$. The sum of four-vectors runs over all the constituents i clustered into the jet. As different heavy resonances have different masses, this feature can be a strong discriminant between jet types. Note that any operation performed on a jet which alter the constituents, such as the pileup mitigation discussed in the next paragraph, may alter the jet mass.

It is important to note that additional proton–proton interactions within a bunch crossing, or pileup, creates additional particles present within an event that can impact jet clustering and the estimation of jet properties. This is especially important for large- R jets which cover large spatial extents. Dedicated pileup removal algorithms are used to mitigate the impact of pileup [17]. Jet trimming [18] is a jet grooming technique used to remove soft and wide angle radiation from jets, in which small radius subjects are removed if they carry a fraction of the jet energy below a threshold. Jet trimming is frequently used on ATLAS to aid in pileup mitigation. The pileup per particle identification algorithm (PUPPI) [19] is frequently used by CMS, in which for each particle a local shape parameter, which probes the collinear vs. soft diffuse structure in the neighborhood of the particle, is calculated. The distribution of this shape parameter per event is used to calculate per particle weights that describe the degree to which particles are pileup-like. Particle four-momenta are then weighted and thus the impact of

(down-weighted) pileup particles on jet clustering is reduce [19]. Pileup mitigation can greatly improve the estimation of the jet mass, energy and momentum by removing/downweighting the pileup particles clustered into a jet that only serve as noise in the jet properties estimation.

Jet identification, energy estimation, and pileup estimation/reduction are among the primary challenges for which the jet images approach has been employed: (i) Jet identification refers to the classification of the parent particle type that gave rise to the jet, and is needed to determine the particle content of a collision event. (ii) Jet energy estimation refers to the regression of the true jet total energy from the noisy detector observations, and is needed to determine the kinematic properties of an event. (iii) Jet pileup estimation and reduction refers to the determination of the stochastic contributions to detector observations arising from incident particles produced in proton–proton collisions that are not from the primary hard scattering. This form of denoising is required to improve the energy and momentum resolutions of measurements of jets.

Among the primary physics settings in which jet images have been used are in studies of jets produced by Lorentz boosted heavy particles, such as a W - or Z -boson, Higgs boson (h), top quark (t), or a hypothetical new beyond the Standard Model particle. When a heavy short-lived particle is produced with a momentum on the order of twice its mass or more, the quark decay products of such a heavy particle have a high likelihood of a collimated emergence in which the subsequent hadronic showers produced by the quarks overlap. Jet clustering algorithms can capture the entirety of the heavy particle decay within one large- R jet with an R -parameter typically between 0.8 and 1.0, though in some cases larger R parameters have been used. The internal structure of such a *boosted jet* can be highly non-trivial and significantly different than a typical jet produced by a single quark or gluon. However, the production of quarks and gluons is ubiquitous at hadron colliders, and thus powerful discrimination methods, or *taggers*, are needed to identify relatively clean samples of heavy-particle-induced boosted jets. Moreover, the mass scale of heavy hadronically decaying particles in the Standard Model

is similar, from the W -boson mass of ~ 80 GeV [20] up to the top quark mass of ~ 173 GeV [20]. Typical discrimination tasks thus include discriminating boosted W -, Z -, h -, or t -jets from quarks and gluons, but also in discriminating between boosted heavy particle jets.

Jet images have also been employed for studying jets from individual quarks and gluons. This includes discriminating between quark and gluon jets, and between jets produced by quarks of different flavor. In these cases, smaller jets typically with $R = 0.4$ are used.

3. Jet Images and Preprocessing

Jet images are built using a fixed grid, or pixelation, of the spatial distribution of energy within a jet. Early instances of such pixelation relied on energy depositions in calorimeter detectors, wherein the angular segmentation of the detector cells was used to define the “pixels” of the jet image and the pixel “intensity” was defined with the transverse energy in a cell. More recently, high resolution measurements of charged particles from tracking detectors have also been used to form images, wherein the transverse momentum of all particles found within the spatial extent of a jet image pixel are summed to define the pixel intensity. While calorimeter and tracking detectors typically span a large angular acceptance, a typical jet has limited angular span. The angular span of a jet is related to the R -parameter of the jet clustering algorithm. Jet images are thus designed to cover the catchment area of the jet [21]. In many cases, the jet image is first defined to be slightly larger than the expected jet catchment area, to ensure that preprocessing steps (discussed in Sec. 3.1) do not disrupt peripheral pixel estimates, and then after pre-processing are cropped. Nonetheless, only a slice of the angular space of the detector is used to define the jet image, with the image centered on the direction of the jet and the image size chosen to capture the extend of a jet with a given R parameter. If depth segmentation is present in a calorimeter, the energy is often summed in depth. From this vantage point, a jet image can be viewed as a gray-scale image comprising the energy measurements encapsulated by the angular

span of the jet. In some cases energy depositions from hadronic and electromagnetic calorimeters will be separated into different images, or separate images will be formed from both calorimeter cell measurements and the spatially pixelated charged particle measurement. In these cases, the set of jet images, each defining a view of the jet from a different set of measurements, can be seen as color channels of jet image.

It should be noted that jet pileup mitigation, such as the aforementioned trimming or PUPPI algorithms, is vital to reduce the impact of pileup on downstream jet image prediction tasks. While not explicitly discussed as a part of the jet image preprocessing, this step is almost always performed prior to jet image formation using the jet constituents, especially in the case of studying large- R jets.

3.1. *Preprocessing*

An important consideration in the training of a classifier is how to process data before feeding it to the classifier such that the classifier can learn most efficiently. For instance, a common preprocessing step in ML is to standardize inputs by scaling and mean shifting each input feature such that each feature has zero mean and unit variance. In this case, standardization helps to ensure that features have similar range and magnitude so that no single feature dominates gradient updates. In general, data preprocessing can help to stabilize the optimization process and can help remove redundancy in the data features to ease the learning of useful representations and improve the learning sample efficiency. However, data preprocessing may come at a cost if the preprocessing step requires approximations that lead to distortion of the information in the data. The primary jet preprocessing steps include:

Translation: An important consideration when preparing inputs to a classifier are the symmetries of data and transformations of inputs that should not affect the classifier prediction. In the case of jet images, these symmetries are related to the physical symmetries of the system. At a particle collider, there is no preferred direction transverse to the beam line, and the physics should be invariant to

azimuthal rotations in the transverse plane. In terms of jet images, given a fixed parent particle, the distribution of jet images at a given azimuthal coordinate $\phi = \phi_a$ should not differ from the distribution at a different $\phi = \phi_b$. As such, an important preprocessing step is to translate all jet images to be “centered” at $\phi = 0$. This is often performed by translating the highest p_T subjet (formed by clustering the jet constituents with a small R -parameter jet algorithms), or the jet energy centroid, to be located at $\phi = 0$. The same invariance is not generically true for changes in η , as translations in η correspond to Lorentz boosts along the beam direction which could alter the jet properties if not handled carefully. When energy is used for jet image pixel intensities, a translation in η while keeping pixel intensities fixed will lead to a change in the jet mass. However, when the transverse momentum, which is invariant to boosts along the beam direction, is used to define pixel intensities, a translation in η can be performed without altering the jet mass distribution. With this definition of pixel intensities, jet images are typically translated such that the leading subjet is located at $\eta = 0$. By centering the jet on the leading p_T subjet, the classifier can focus on learning the relative variations of a jet, which are key for classification.

Rotation: The radiation within a jet is also approximately symmetric about the jet axis in the η - ϕ plane. A common preprocessing step is thus to rotate jet images, after centering the image on the leading p_T subjet, such that the second leading p_T subjet or the first principle axis of spatial distribution of p_T in the image is aligned along the y -axis of the image. However, there are challenges with rotations. First, rotations in the η - ϕ plane can alter the jet mass, thus potentially impacting the classification performance.^a Second, as jet images are discretized along the spatial dimensions, rotations by angles other than factors of $\pi/2$ cannot be performed exactly. One approach is to perform a spline interpolation of the p_T distribution within a jet image, apply a rotation to this spline function, and then impose an image grid to discretize the spline back to an image.

^aAlternative definitions of rotations have been proposed that preserve jet mass [22] but may alter other key jet properties.

The interpolation and the post-rotation discretization can spatially smear information in the jet and lead to aliasing. As such, there is varying use of rotation preprocessing in jet image research.

Flipping: A transformation $\phi \rightarrow -\phi$ should not affect the physics of the jet, and this transformation can be performed to ensure that positive ϕ contains the half of the jet with more energy, for instance due to radiation emission.

Normalization: A step often found in image preprocessing for computer vision tasks is image normalization, typically through taking an L^2 -norm of the image such that $x_i \rightarrow x_i / \sum_j x_j^2$ where x_i is a pixel intensity and the sum runs over all pixels in an image. However, in the case of jet images, such a normalization may be destructive, as it does not preserve the total mass of the jet (as computed from the pixels) and can deteriorate discrimination performance due to this loss of information [23]. As such, there is varying usage of image normalization in jet image research.

The impact on the jet mass, as computed from the pixels of jet images, for W -boson jets within a fixed p_T range and within a fixed pre-pixelation mass range can be found in Fig. 2. The distortion on the jet mass from pixelation, rotations for images with energies as pixel intensities, and from L^2 normalization, can be seen clearly, whilst translation and flipping do not show distortions of the jet image mass. As expected, mild distortion of the mass can be seen when rotations are performed on jet images with transverse energy used for pixel intensities. These distortions may or may not be impactful on downstream tasks, depending on if the jet mass is a key learned feature for the downstream model.

4. Computer Vision and Convolutional Neural Networks

Object classification in computer vision (CV) tasks served as a primary setting where deep learning had major early successes [24], quickly surpassing then state-of-the-art approaches and serving as one of the drivers for a deep learning revolution. While much of

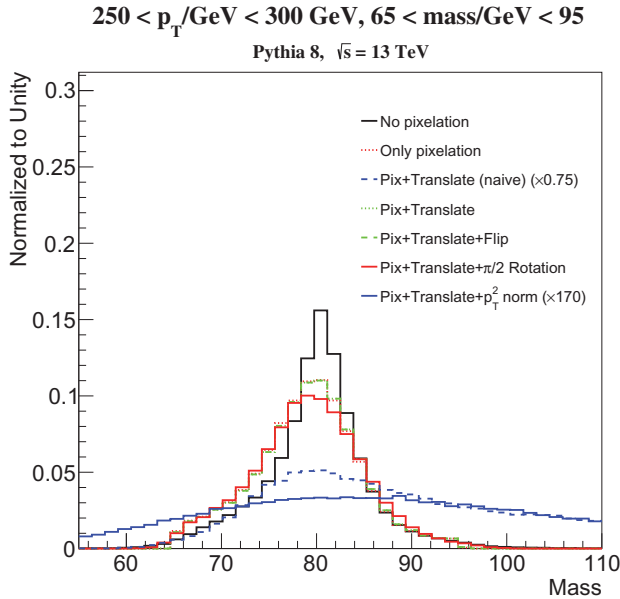


Fig. 2. The impact of various preprocessing steps on the distribution of estimated jet mass for boosted W -boson jet images [23].

the work in CV has focused on understanding natural images, data collected by physics experiments come from heterogeneous detectors, tend to be sparse, and do not have a clear similarity to natural images. Nonetheless, the success of DL in CV inspired a parallel effort in the collider physics community to explore applications of such techniques to HEP data. Below we present a brief introduction to convolutional neural networks (CNNs) [25] and some of the state-of-the-art architecture variants in order to provide some background for the models used in jet tagging and other applications. For a more in depth pedagogical introduction to this material, see for instance [26].

Most of the models discussed in this text rely on the use of convolutional layers. However, it should be noted that some models make use of locally-connected layers [25, 27, 28], in which a given neuron only has access to a small patch of an input but, unlike convolutional layers that rely on weight sharing (as discussed below), the

neuron processing each image patch is associated with a different set of weights.

Convolutional neural networks rely on neuron local spatial connectivity and weight sharing to produce translationally equivariant models that are well adapted to image analysis. A typical CNN is built by stacking one or more convolutional and nonlinear activation layers often followed by a pooling layer. This structure is repeated several times. Fully connected layers, with full connections from all inputs to activations, are used to perform the final classification or regression prediction. Images processed by CNNs are represented as 3D tensors with dimensions $width \times height \times depth$ and are often referred to as the image volume. The height and width dimensions correspond to the spatial extent of the image while the depth is typically the color channel.

Convolutional layers are composed of a set of *filters*, where each filter applies an inner product between a set of weights and small patch of an input image. The filter is scanned, or *convolved*, across the height and width of the image to produce a 2D map, often referred to as a response map or convolved image, that gives the response of applying the filter at each position of the image. The response at each position becomes large when the filter and the image patch match, i.e. when their inner product is large. The filters will thus learn to recognize visual features such as edges, textures, and shapes, and produce large responses when such visual features are present in a patch of an image. The spatial extent of the input patch is known as the receptive field or filter size, and the filters extend to the full depth of the image volume. Several filters are learned simultaneously to respond to different visual features. The response map of the filters are then stacked in depth, producing an output convolved image volume. Finally, the response maps are passed through point-wise (i.e. per pixel) nonlinear activations to produce an activation map.

By sharing weights between neurons, i.e. by scanning and applying the same filter at each image location, it is implicitly assumed that it is useful to apply the same set of weights to different image locations. This assumption is reasonable, as a visual feature may be

present at any location in an image and the filter is thus testing for that feature across the image. This results in the convolutional layers being translationally equivariant, in that if a visual feature is shifted in an image, the response to that feature will be shifted in the activation map. In addition, parameter sharing results in dramatic reduction in the number of free parameters in the network relative to a fully connected network of the same number of neurons.

Pooling layers reduce the spatial extent of the image volume while leaving the depth of the volume unchanged [29]. This further reduces the number of parameters needed by the network and helps control for overfitting. Pooling is often performed with a *max* operation wherein only the largest activation in a region, typically 2×2 , is kept for subsequent processing.

Normalization layers may be used to adjust activation outputs, typically to control the mean and variance of the activation distribution. This helps ensure that a neuron does not produce extremely large or small activations relative to other neurons, which can aid in gradient-based optimization and in mitigating exploding/vanishing gradients. Batch normalization [30] is a common normalization method in which, for each mini-batch, the mean and variance within the mini-batch of each activation dimension are used to normalize the activation to have approximately zero mean and unit variance. A linear transformation of the normalized activation, with learnable scale and offset parameters, is then applied.

Fully connected layers are applied at the end of the network after *flattening* the image volume into a vector in order to perform classification or regression predictions. At this stage, auxiliary information, potentially processed by a separate set of neural network layers, may be merged with the information gleaned from the processing by convolutional layers in order to ensure that certain features are provided for classification. Within the jet tagging context, such information may correspond to information about the jet, such as its mass, or global event information such as the number of interactions in a given collision.

Residual Connections: While CNNs encode powerful structural information into the model, such as translation equivariance, it has been noted that scaling up such models by stacking large numbers of convolutional layers can lead to large challenges in training [31]. In order to train large models using the backpropagation algorithm, the chain rule is used to compute the gradient from the model output back to the relevant weight. In early layers, the multiplication of many gradients can lead to vanishingly small or exploding gradients, thus resulting in unhelpful gradient updates. To overcome this challenge, the residual block [32] was proposed, and has led to the development of *residual networks*. While a typical neural network layer passes input z through a nonlinear function $f(\cdot)$ to produce an output $z' = f(z)$, a residual block also uses a “skip connection” to pass the input to the output in the form $z'_{\text{res}} = W_s z + f(z)$ where the weights W_s can be used to project the channels z to have the same dimension as the function $f(z)$. In this way, the function $f(\cdot)$ is tasked with learning the relative change to the input. Moreover, the skip connection provides a path to efficiently pass gradients backwards to earlier layers of the network without distortion through the nonlinearities, making gradient descent much easier and thus enabling the training of significantly deeper models. Note that the function $f(\cdot)$ can contain several layers of convolutions, nonlinearities, and normalization before being recombined with the input.

Training in supervised learning tasks is performed by minimizing an appropriate loss function that compares the CNN prediction with a true label. The loss is typically the cross-entropy in the case of binary classification, and the mean squared error in the case of regression. Minimization is performed using stochastic gradient descent (SGD) [33], or one of its variants such as ADAM [34] designed to improve the convergence of the learning algorithm.

Evaluation Metrics: Receiver operating characteristic (ROC) curves are frequently used to examine and compare the performance of binary classification algorithms. Given a model which produces a classification prediction $c(x)$, where x is the input features and

$c(\cdot) \in [0, 1]$, the model is applied to a set of inputs thus yielding a distribution of predictions. A threshold τ on the prediction is scanned from 0 to 1, and the fraction of inputs for each the signal and background classes above this threshold, i.e. the signal efficiency (ϵ_S) and background efficiency (ϵ_B) for surviving this threshold, defines a point on the ROC curve for each τ value. ROC curves thus display the background efficiency (or background rejection defined as 1 divided by the background efficiency) vs. the signal efficiency. When the ROC curve is defined as the background efficiency vs. the signal efficiency, a metric commonly used to evaluate the overall model performance is the ROC integral, also known as the area under the curve (AUC).

Significance improvement characteristic (SIC) curves [35] are closely related to ROC curves, but display $\epsilon_S/\sqrt{\epsilon_B}$ as a function of the signal efficiency ϵ_S . This curve targets displaying the potential improvement in statistical significance when applying a given discriminant threshold relative to not applying such a threshold.

5. Jet Tagging

Jet tagging refers to the classification of the parent particle which gave rise to a jet. Linear discriminant methods were first applied to jet images defined using a single channel, or “gray-scale”, with pixel intensities defined as the calorimeter cell p_T [1]. Subsequently, CNN-based classifiers trained on single-channel images were developed for discriminating between W/Z jets and quark/gluon jets [23, 27], between top jets and quark/gluon jets [10, 36–38], and for discriminating between quarks and gluons [39]. Quark/gluon discrimination with single-channel jet images has also been explored for use in heavy ion collisions [40]. The extension to utilizing jet images “in color” with multiple channels, defined for instance using charged particle information, has shown promising performance improvements over single-channel approaches in many of these tasks [10, 39, 41–45], and has been explored in realistic experimental settings by the ATLAS and CMS collaborations [46, 47].

5.1. Jet tagging on single-channel jet images

W/Z Tagging: The discrimination of boosted W and Z vector boson initiated jets from quark/gluon jets has served as a benchmark task in boosted jet tagging. The color singlet nature of electroweak bosons decaying to quark pairs leads to an internal structure of boosted W/Z jets in which there are typically two high-energy clusters, or subjets, and additional (dipole) radiation tends to appear in the region between such subjets. The Higgs boson, also a color singlet with decays to quark pairs, has a similar substructure, although the decays of heavy flavor bottom and charm quark pairs can lead to some structural differences owing to the long lifetime of such quarks and their harder fragmentation than lighter quarks. In contrast, single quarks and gluons tend to produce jets with a high-energy core, lower energy secondary subjets created through radiative processes, as well as diffuse wide angle radiation further from the core of the jet. These features can be seen clearly in Fig. 3, which shows the average W -boson jet image and average quark/gluon jet image after preprocessing.

Building ML models applied to jet images for this discrimination task avoids the explicit design of physics-inspired features, and

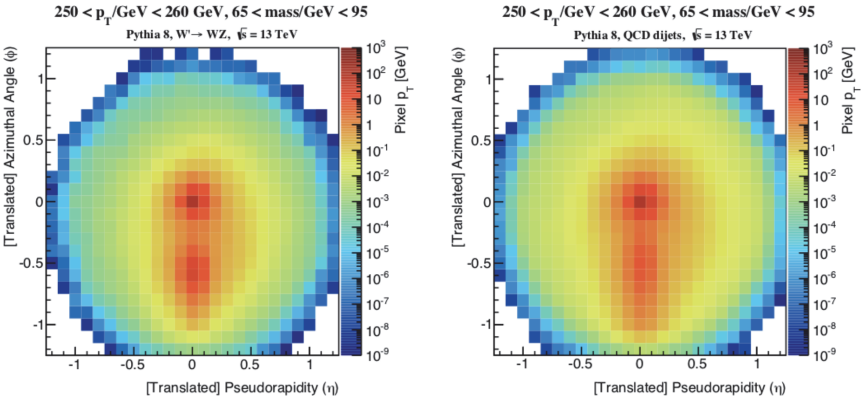


Fig. 3. Average W -boson jet image (left) and average quark/gluon jet image (right) after preprocessing [23].

rather focuses on the learning task of identifying differences in the jet image spatial energy distributions. In phenomenological studies, both fully convolutional [23] and models with locally connected layers [27] have been examined for discriminating jet images of boosted W and Z vector boson initiated jets from quark/gluon jets. The CNN models were examined in simulated samples of jets without pileup. The locally connected models were examined in events both with and without pileup, thus enabling the examination of the impact of pileup noise on jet image-based tagging.

Within both the studies on convolutional [23] and locally connected [27] models, hyperparameter scans were performed to find model parameters that maximized performance.^b The hyperparameters that were considered in the scans included the number of convolutional/locally connected layers, the number of hidden units per layer, and the number of fully connected layers. The resulting optimized models were similar, containing 3–4 convolutional or locally connected layers, as well as 2–4 fully connected layers with approximately 300–400 hidden units at the end of the network. In the CNN, 32 filters were used in each convolutional layer, as well as (2×2) or (3×3) downsampling after each convolutional layer. One notable additional optimization performed for the CNN models was the size of the convolution filters in the first layer. While filter sizes are typically (3×3) or (4×4) in standard CV applications, in the case of application to jet images it was found that a larger (11×11) filter in the first convolutional layer (with later layers using standard (3×3) filter sizes) resulted in the best performance. It was hypothesized that the such large filters were beneficial when applied to sparse images [23], in order to ensure that some non-zero pixels are likely to be found within the image patch supporting the filter application.

The ROC curves indicating the performance of the CNN model and locally connected model (applied to jets with pileup included) are shown in Fig. 4. It should be noted that the jets in these figures

^bIn the case of CNNs the AUC was maximized whilst the Spearmint Bayesian Optimization package [48] was used to optimize the model with locally connected layers.

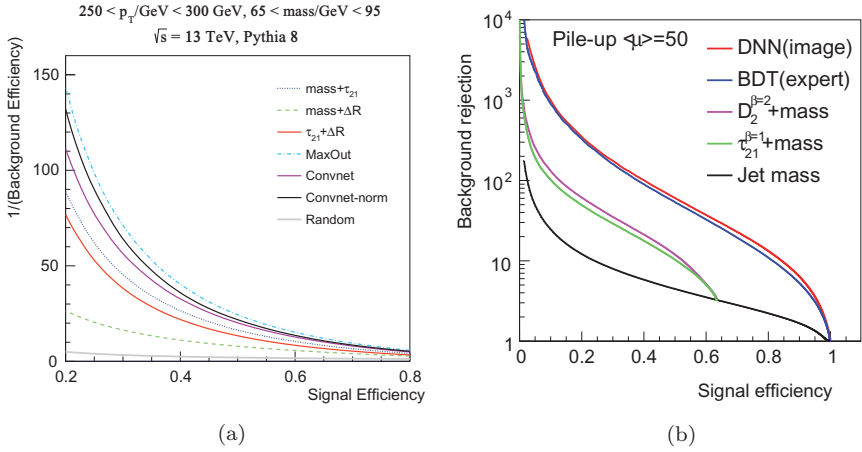


Fig. 4. ROC curves for quark/gluon background rejection vs. boosted W boson tagging efficiency for (a) events without pileup [23], and (b) events with pileup [27]. The jet image-based CNN taggers are seen to outperform combinations of jet substructure features, and to be stable with respect to the addition of pileup.

correspond to different p_T ranges, with jets of $p_T \in (250, 300)$ GeV for the CNN model and of $p_T \in [300, 400]$ GeV for the locally connected model, and thus are not directly comparable. Also shown are combinations of common physics expert engineered jet substructure features, such as the jet mass, the distance between the two leading p_T subjets, the τ_{21} n -subjettiness [49], and the energy correlation function $D_2^{\beta=2}$ [50]. Two variable combinations were computed using 2D binned likelihood ratios. Both the CNN and locally connected model significantly outperform the 2D jet substructure feature combinations. It can also be seen that the jet image approach is not overly sensitive to the effects of pileup as the large performance gain over jet substructure features persists both with and without the presence of pileup, owing to the use of jet trimming to reduce the impact of pileup noise in the jets. In addition, a boosted decision tree (BDT) classifier [51] combining six substructure features was compared with the locally connected model and found to have similar performance. While these early jet image-based models did not significantly outperform combinations of several jet substructure features, this may

be due to their relatively small model structure. As will be seen, more complex architectures and the use of multi-channel jet images can lead to large gains over combinations of jet substructure features.

One can also see the effect of L^2 image normalization on CNN models, which appears to improve performance over unnormalized images. This effect was found to occur because the CNN model output was observed to have only small correlation with the jet mass, and thus was not learning to be heavily reliant on the jet mass information that is distorted by normalization. As a result, the regulation of the image variations due to normalization was found to be beneficial enough to overcome the induced distortion of the jet mass. With more powerful models that learn representations more correlated to the jet mass, this balance may not occur.

Top Tagging: The discrimination between boosted top quark jets and quark/gluon jets using CNNs applied to jet images has also been examined both in phenomenological studies [36] and in realistic simulations by the CMS experiment [47]. Top quark jet images are structurally more complex than $W/Z/h$ jet images as hadronic decays of top quarks contain three quarks. This can have implications on both the preprocessing and the tagging performance. That is, some of the pre-processing steps previously defined will lead to uniformity among jet images for two quark systems, such as the rotation step which aligns the leading two subjets, but may not lead to the same level of uniformity for three quark systems.

The DeepTop [36] model is a CNN applied to single-channel jet images after the preprocessing described above, including image normalization. Hyperparameter optimization yielded a model with four convolutional layers, each with eight filters of size (4×4) , MaxPooling for image downsampling after the second convolutional layer, and three dense layers of 64 hidden units each for classification. For these phenomenological studies, the model was trained with approximately 150k jets using the mean squared error (MSE) loss. While structurally similar to the single-channel CNN used for W/Z tagging in reference [23] there are some notable differences such as the use of fewer numbers of filters (8 rather than 32) and the smaller filter size

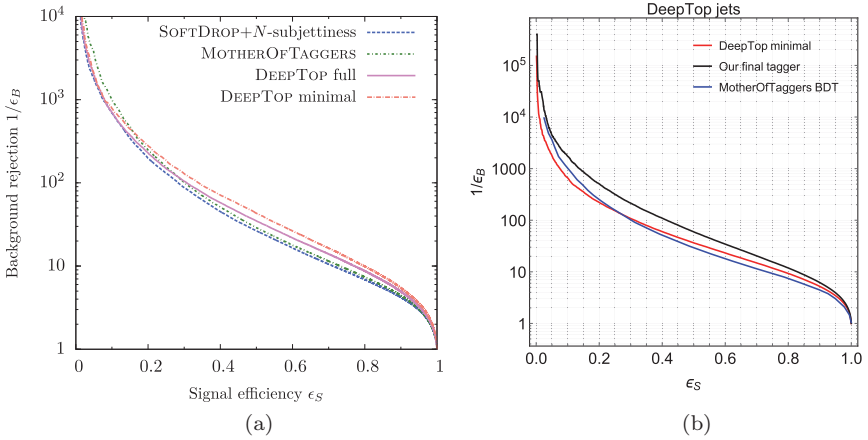


Fig. 5. ROC curves for quark/gluon jet rejection vs. boosted top efficiency for (a) the DeepTop model [36], and (b) the updated DeepTop model from [42]. In both cases, the CNN-based DeepTop models outperform individual and BDT combinations of substructure features, while the updated model in (b) is also seen to significantly improve the DeepTop performance.

in the first layer of convolution. The reason for these difference may be due to (a) the presence of three quarks in the top quark decay leads to more pixel-populated images and thus allowed for the use of smaller initial filter sizes, or (b) the global nature of the hyperparameter scan wherein the number of filters and the size of the filters was fixed to be the same across all convolutional layers.

The performance of the DeepTop model can be found in Fig. 5(a) in terms of the ROC curve comparing the quark/gluon rejection vs. the boosted top jet tagging efficiency for jets with $p_T \in [350, 450]$ GeV. In this momentum range, the decay products of the top quark may not be contained in a single jet, and such a containment was not required for the jets under study. DeepTop was compared with a combination of mass and n -subjettiness, as well and a BDT, denoted MotherOfTaggers, combining several jet substructure features. The jet image-based DeepTop algorithm showed clear performance gains over substructure approaches across most of the signal efficiency range. As previously mentioned, pre-processing steps have the potential to be beneficial for the learning process by

producing more uniform images, but may also lead to performance degradation. This was studied within the scope of the DeepTop algorithm, by examining the tagging performance using full preprocessing and a minimal preprocessing that only performed centering but not the rotation or the flipping. This can be seen in Fig. 5(a), where a clear performance benefit was observed when utilizing only minimal pre-processing. While the full pre-processing may be beneficial for small sample sizes, with sufficient sample sizes and model complexity the CNN models appear able to learn well all the variations in jet images. In this case, the approximations introduced by pre-processing steps appear to be more detrimental than the benefits from uniformization of the jet image distributions.

Building upon the DeepTop design, developments in architecture design, jet image preprocessing, and optimization were introduced in the phenomenological study of [42]. These developments include: (i) the cross entropy loss function, rather than the mean squared error loss, was used as it is more suitable to binary classification problems, (ii) a learning rate adaptive optimizer, AdaDelta [52], and small mini-batch sizes of 128 was used rather than vanilla stochastic gradient descent and large mini-batches of 1000, (iii) larger numbers of filters per convolutional layer, between 64 and 128 rather than 8, and 256 neurons in the dense layers instead of 64, (iv) preprocessing is performed before pixelation under the assumption that one would have access to high resolution particle momentum measurements, for instance using Particle Flow [14] approaches to jet reconstruction, and (v) the training set size was increased by nearly a factor of 10. While the individual effects of these developments will be examined further in Sec. 5.2 when discussing top tagging on multi-channel jet images, the combination of these developments can be seen to provide large performance improvements over DeepTop of nearly a factor of two in background rejection at fixed signal efficiencies in Fig. 5(b).

In terms of more complex architectures, the ResNeXt-50 architecture [53] was adapted to boosted top jet tagging task using single-channel jet images in the phenomenological studies in [10]. ResNeXt-50 utilizes blocks containing parallel convolutional layers that are aggregated and merged also with a residual connection at

the end of the block. As the jet images typically have fewer pixels than natural images, the architecture was adapted to the top tagging dataset by reducing the number of filters by a factor of four in all but the first convolutional layer, and dropout was added before the fully connected layer. In addition, smaller pixel sizes in the jet images were utilized in this model, with a granularity of 0.025 radians in η - ϕ space (whereas the jet image granularities typically used in other models is 0.1 radians in η - ϕ space).

The ROC curve comparing the ResNeXt-50 model to a CNN based on [36, 42], and comparing to several other neural network models with varying architectures can be found in Fig. 6. The ResNeXt-50 model provides approximately 20% improvement in background rejection for fixed signal efficiency over the CNN model, and is among the most performant algorithms explored. This is notable as many of the other neural network models utilize particle 4-vectors as inputs, rather than aggregated particle information

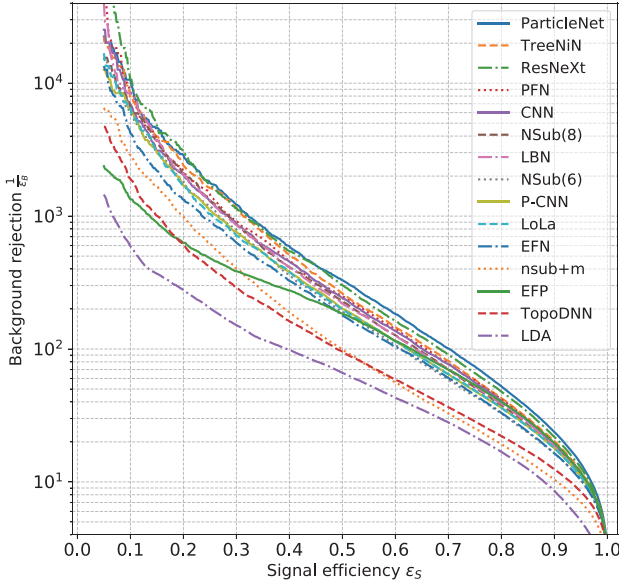


Fig. 6. ROC curve comparisons of various boosted top tagging models is shown [10]. Both ResNeXt and CNN curves are jet image-based taggers using CNN-based architectures.

with a pixel cell, and make use of particle charge information, while the ResNeXt model only utilizes the distribution of energy within the jet. However, the ResNeXt-50 model contains nearly 1.5 million parameters, which is far more than other models such as the CNN which contains ≈ 610 k parameters and the tree structured neural network (TreeNiN) which contains ≈ 34 k parameters. Thus powerful information for discrimination can be extracted with jet image-based models even from single-channel images, but it may come with the price of models with large parameter counts.

This model comparison study has been performed in a phenomenological setting on particle level simulations, and the ultimate question remains as to the suitability for using these models in real experiment settings. In experimental settings, realistic detector noise, detection efficiency, detector heterogeneity, and data taking conditions such as pileup, underlying event, and beam evolution will impact the model performance. Powerful models, including the large ResNext and CNN models, will likely have sufficient flexibility to learn powerful discriminators even in these more challenging settings. However, in general it remains to be seen if these models can be accurate whilst maintaining a low calibration error (where calibration in this context refers to the criteria that the predicted class probabilities correspond to the true probabilities of a given data input having a given label) [54], or if additional care is needed to ensure calibration. Moreover, applications in real experimental settings must consider systematic uncertainties associated with training ML models in (high fidelity) simulation but applying them in real data with potentially different feature distributions. The relationship between model complexity and sensitivity to systematic uncertainties in real experiment settings still remains to be thoroughly explored. The potential benefits in terms of sensitivity to systematic uncertainties when using neural networks with different structural assumptions, such as convolutional vs. graph models, also requires further study and will likely depend on the details of how a given systematic uncertainty effects the feature distributions. Some exploration of these challenges can be found in Sec. 5.3 examining model sensitivity to theoretical uncertainties and in Sec. 5.4 examining applications of these models in

HEP experiments. Nonetheless, these remain important and exciting avenues of future work.

Decorrelated tagging with Jet Images: A common strategy in HEP to search for a particle is the so-called *bump hunt* in which the particle would give rise to a localized excess on top of a smoothly falling background in the distribution of the mass of reconstructed particle candidates. For instance, one may aim to identify the W -boson mass peak over the quark and gluon background from the distribution of jet mass. In addition to the particle mass being localization, a key to this strategy is that the smoothly falling background mass distribution can typically be characterized with simple parametric functions, thus facilitating a fit of the data to identify the excess above this background. Jet classification methods can cause challenges in the aforementioned strategy, as the classifier may preferentially select for jets with a specific mass, thereby sculpting the selected jet mass distribution of the background and rendering the search strategy unusable. As a result, one line of work has focused on de-correlating classifiers from a sensitive feature (e.g. mass) such that the sensitive feature is not sculpted by the application of the tagger. Such methods tend to rely on data augmentation or regularization, and overviews of these methods can be found for instance in [55, 56]. Two recent regularization techniques that have seen strong de-correlation capability include (i) adversarial techniques [57, 58], wherein a second neural network is trained simultaneously with the jet classifier to penalize the jet classifier when the value of the sensitive feature can be predicted from the classifier’s output or its hidden representations, and (ii) distance correlation regularizers [59], wherein the jet classifier loss is augmented with an additional regularization which explicitly computes the correlation between the classifier predictions and the sensitive feature. In both cases, the amount of penalization from the regularization can be varied through a hyperparameter scaling the relative size of the regulation term to the classification loss.

De-correlation for W -boson jet tagging with jet images using CNNs was examined in phenomenological studies in [59], using a

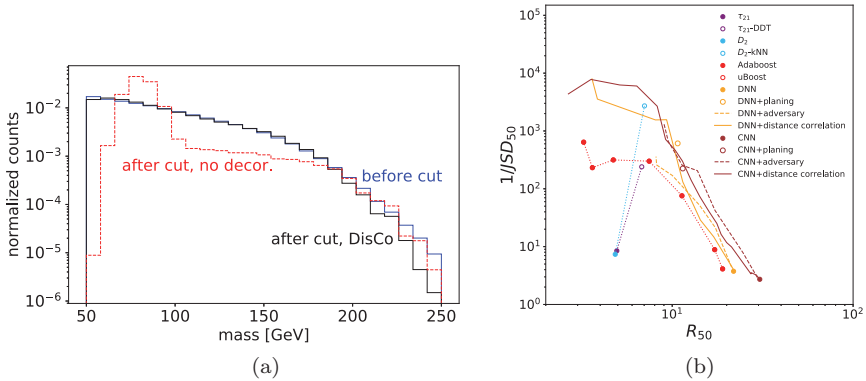


Fig. 7. (a) For boosted W -boson tagging, the jet mass is shown before applying a threshold on a trained CNN tagger and after applying a threshold on a standard and mass decorrelated tagger [59]. A clear reduction in mass sculpting is observed. (b) The rejection at 50% signal efficiency vs. one over the Jensen–Shannon divergence, computed on the binned jet mass distribution before and after tagging, is shown for various taggers [59]. Jet image-based CNN taggers are seen to outperform other methods, either using adversarial or distance-correlation-based mass decorrelation.

CNN architecture similar to the model described in [42]. The quark and gluon background jet mass distribution before and after applying a threshold on the output of a CNN can be seen in Fig. 7(a), showing a clear sculpting of the mass distribution. However, when the distance correlation regularization, or *Disco*, is used during training, the mass distribution remains largely unsculpted after applying a classification threshold. The level of de-correlation can be estimated by examining the agreement between the mass distribution before and after applying a classifier threshold, for instance using the Jensen–Shannon divergence (JSD) computed between the binned mass distributions. For classifier thresholds fixed to 50% signal efficiency, Fig. 7(b) shows the JSD as a function of the background rejection where the curves are produced through training with varying sizes of the regularization hyperparameter. The CNN models are compared with neural networks trained on substructure features and other classifiers with de-correlation methods applied. The CNN models, both the adversarial and distance correlation regularization, are

seen to typically provide the highest background rejection for a given level of de-correlation compared to other models.

5.2. *Multi-channel jet tagging with CNNs*

Recent work on jet image-based tagging has shown performance gains through the use of multi-channel images. While single-channel jet images have provided gains in classification performance over individual, or pairings of, engineered substructure features, the performance benefits were typically smaller when compared to ML models trained on larger groups of substructure features (except when very large models were used, as in [10]). Multi-channel jet images use calorimeter images as only a single input image channel, with additional channels computed from charged particle features such as the momentum, multiplicity, or charge. There is a significant amount of freedom in choosing the definition of the additional image channels, allowing for a flexibility in the choice of inductive bias to deliver relevant information to the CNN.

One challenge in combining charged particle trajectory information and calorimeter images is the mismatch in resolution; charged particle trajectories tend to have a significantly finer spatial resolution than calorimeters, thus leading to the questions of how to combine such information. As charged particles are not measured on a regular grid, often the same spatial grid for the calorimeter component is used for the charged particle image and the energy of the constituents is summed within each pixel. Alternatively, separate CNN blocks (or upsampling procedures) can be used to process charged and calorimeter images separately into a latent representation of equal size such that they can be merged for further processing. Note that when particle flow objects are used, and thus both neutral and charged particle measurements do not necessarily fall on a grid, a fine grid can be used to exploit the better charged particle momentum resolution. It should also be noted that while phenomenological studies at particle-level often use fixed grids to emulate the discretization of real detectors, different inputs (i.e. charge vs. neutral) in real detector settings have different resolutions which may be difficult to account for in simple discretization approaches.

Multi-channel jet image-based tagging was introduced in phenomenological studies of discriminating between quark initiated and gluon initiated jets [39, 41] and has since been explored within the *quark vs. gluon* context on the ATLAS experiment [46], in CMS Open Data [60, 61], and for tagging in heavy-ion collision environments [40]. More broadly, multi-channel jet image tagging has lead to improved performance in phenomenological studies of boosted top quark jet tagging [10, 37, 42], as well as in boosted W/Z jet tagging [43] and in boosted Higgs boson tagging [44, 45]. Notably, multi-channel jet image-based boosted top tagging has been explored on the CMS experiment [47] including the comparison and calibration of this discriminant with respect to CMS collision data, thus adding additional insights into the usability of such models within LHC data analysis.

The use of multi-channel jet images built from charged particle momentum and multiplicity information within the context of discriminating between quarks and gluons is natural, as the number of charged particles within such a jet is known to be a powerful discriminant for this challenging task [62]. As such, in the phenomenological studies of [39] three jet image channels were defined: (1) the transverse momentum of charged particles within each pixel, (2) the transverse momentum of neutral particles within each pixel, and (3) the charged particle multiplicity within each pixel. The same pixel size was used in each image, thus facilitating the direct application of multi-channel CNNs. This approach thus relies on the ability to separate the charged and neutral components of a jet; while the charged component is measured using tracking detectors, the unique identification of the neutral component of a jet is significantly more challenging task. However, advancements in particle flow [14] aid in such a separation, albeit not perfectly and with differing resolutions between charged and neutral measurements.

The benefit of the multi-channel approach for quark vs. gluon discrimination can be seen in the ROC and SIC curves in Fig. 8. Both the calorimeter only approach, denoted Deep CNN grayscale, as well as the multi-channel approach, denoted Deep CNN w/color, outperform single features engineered for this task, BDTs trained using five of such features, and a linear discriminant trained on the

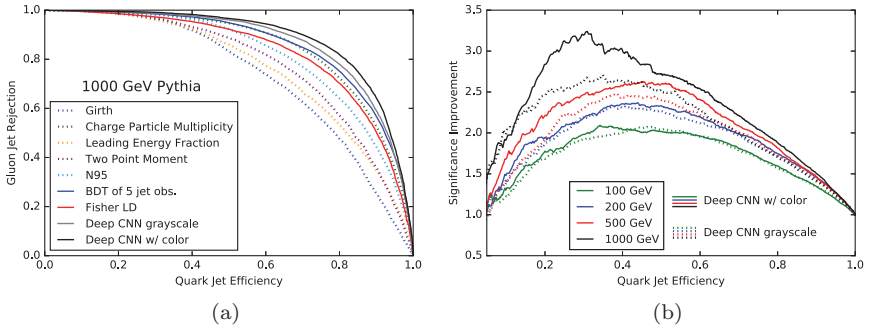


Fig. 8. ROC curve (a) and SIC curve (b) for quark vs. gluon tagging using multi-channel jet images [39]. Comparisons with jet substructure-based discriminants is shown in (a), while comparison between single-channel and multi-channel jet image-based tagging with CNNs is shown in (b).

grayscale jet images. In addition, the multi-channel model is seen to dominate over the single-channel model in both the ROC curve for jets with a momentum of $p_T \approx 1000$ GeV and in the SIC curve across a range of jet momentum. The multi-channel approach is found to be especially beneficial at higher momentum where the jets have a large charged particle multiplicity.

This multi-channel approach using charged, neutral, and multiplicity channels was also found to be powerful in phenomenological studies of discriminating between boosted Higgs boson jets and a background of gluon splitting to $b\bar{b}$ jets in multi-jet events [44]. In addition to a CNN focused on discrimination based on jet images, this work also explored simultaneously processing an *event image*, defined using the aforementioned three channels over the entire calorimeter, through a separate set of convolutional layers and combining with the output of the convolutional processing of jet image before discrimination. By including such an event image, one may explore the potential benefits of event topology information outside of the jet image for discrimination. The SIC curve for this discrimination task can be seen in Fig. 9, where the CNN approaches were seen to significantly outperform single engineered features. CNNs using only the jet image, event image, or both (denoted “Full CNN Architecture” in Fig. 9) were compared, showing that much of the discrimination

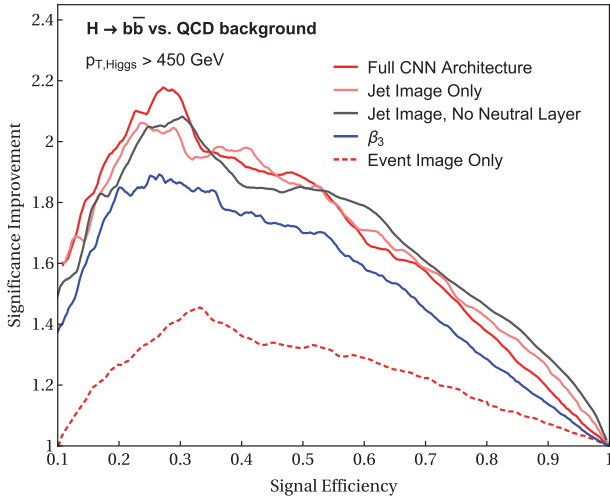


Fig. 9. SIC curve for boosted Higgs to $b\bar{b}$ tagging using multi-channel jet images [44]. Models using only jet images, and models using both jet images and “event images” are shown.

power rests in the jet image whilst the event image may provide some modest improvements. In addition, the jet image discrimination without the neutral particle channel was also found to be comparable to one using the neutral channel, indicating that much of the discrimination power lies in the charged particle information within the jet.

While a clear approach to extending jet images to contain multiple channels is to sum the momentum of the charged particles or compute multiplicities in each pixel to form an image channel, the high resolution of the charged particle information allows for the introduction of additional inductive bias. More specifically, given the set of charged particles contained in the region of a pixel, one may compute pixel-level features that may be more amenable to a given discrimination task. This approach was followed for building CNNs to discriminate between (a) up and down type quarks, and (b) quarks and gluons [41]. In these phenomenological studies, knowledge of the utility of the jet charge feature [63–65] for discriminating jets of different parent particle charge inspired the development of the jet image channel computed per pixel as the p_T weighted charge

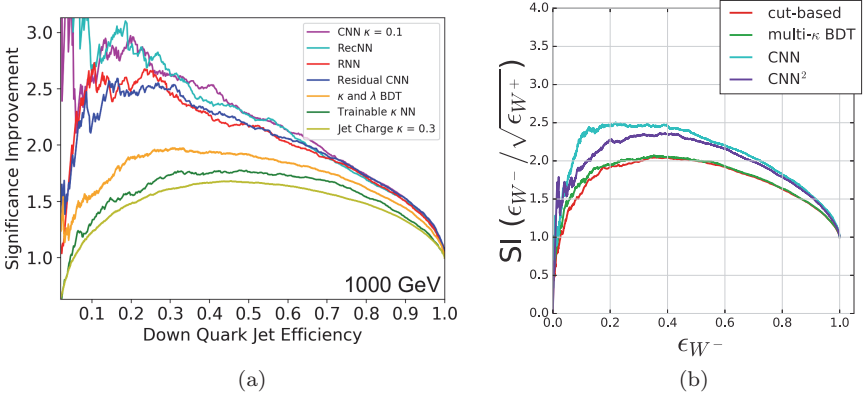


Fig. 10. SIC curves for (a) discriminating down quarks from up quarks [41], and (b) discriminating between W^+ and W^- bosons [43]. The CNN $\kappa = 0.1$ model in the left figure and the CNN models of the right figure utilize the per pixel p_T weighted charge image.

$Q_\kappa = \frac{1}{(\sum_j p_T^{(j)})^\kappa} \sum_j Q^{(j)} (p_T^{(j)})^\kappa$. The SIC curve showing the performance of the CNN trained on the two channel jet images, one channel for p_T and one for Q_κ per pixel, is shown in Fig. 10(a). The two channel CNN significantly outperformed the total jet charge and classifiers trained on engineered features, and is comparable to other deep architectures trained for this task.

A similar jet charge-based multi-channel CNN was explored for discriminating between boosted $W^+/W^-/Z$ boson jets in the phenomenological studies of [43]. The per pixel charge image averaged over the test set for W^+ , W^- , and Z jet images is shown in Fig. 11. The geometry of all three images is similar, but the average per pixel charge differs significantly as expected, with the typical W^+ image carrying a positive pixel value, the typical W^- image carrying a negative pixel value, and the typical Z image having charge close to zero. The SIC curve for discriminating between W^+ and W^- jets can be seen in Fig. 10(b). Two CNNs were explored in this work, one denoted *CNN* in which both a p_T and Q_κ image were processed together (i.e. as a single multi-channel image processed by convolutional layers) and one denoted *CNN*² in which each channel is processed by a separate stack of convolutional layers and then combined

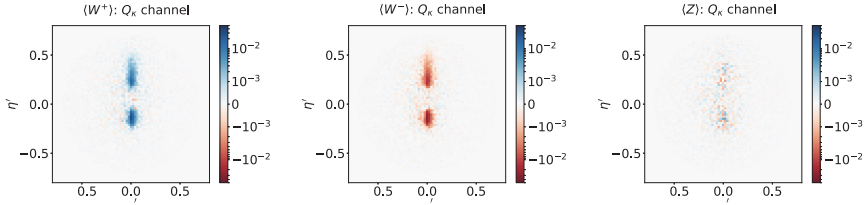


Fig. 11. Average image of the per pixel p_T weighted charge Q_κ is shown for W^+ (left), W^- (middle), and Z -bosons (right) [43].

before the classification layers. Both CNNs significantly outperform methods based on engineered features.

Multi-channel jet images were explored for top tagging in the phenomenological studies of [42], using four-channel jet images defined with the neutral jet component as measured by the calorimeter, the charged particle sum p_T per pixel, the charged particle multiplicity per pixel, and the muon multiplicity per pixel. The architecture is discussed in Sec. 5.1 within the context of single-channel jet images. The inclusion of the muon image channel targets the identification of b quark initiated subjets within the top jet as muons can be produced in b -hadron decays. As noted in Sec. 5.1, several changes to the model architecture, preprocessing, and training procedure relative to the first proposed DeepTop model [36] were included in this work. The impact of these individual changes can be seen in Fig. 12 wherein developments on top of the first proposed DeepTop model [36] are sequentially added to the model and the resulting ROC curve is shown. The inclusion of multiple “color” channels was only seen to provide modest performance gains over single-channel jet images. Notable among changes that led to the largest improvements were changing the optimization objective to be more suitable for classification tasks and changing the optimizer to ADAM (denoted training in the figure), increasing the model size (denoted architecture in the figure), and increasing the sample size. In agreement with these results, recent CNN models built for processing jet images have also tended to focus on larger models with large samples for training.

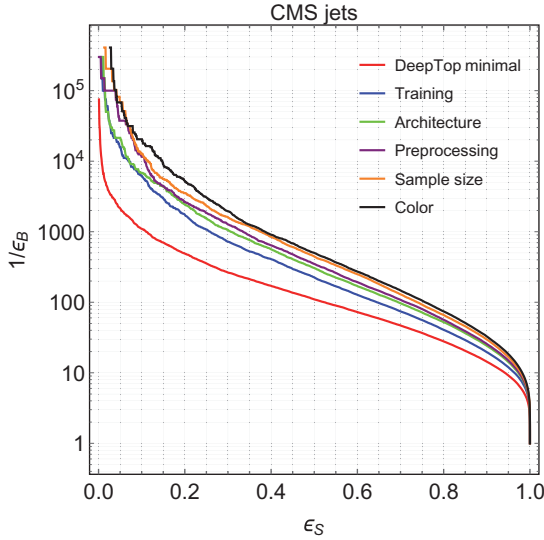


Fig. 12. ROC curve of boosted top jet tagging efficiency vs. background quark and gluon rejection for the minimal DeepTop model [36] compared with models sequentially including the changes proposed in [42].

5.3. Sensitivity to theory uncertainties

While matrix element and parton shower Monte Carlo generators often provide high fidelity predictions of the data generation process, they provide only approximations to the scattering and showering processes and empirical models of the hadronization process. As such, uncertainties in the theoretical predictions of these generators must be propagated to downstream analyses. One mechanism for doing this is to compare an observable computed with samples from different Monte Carlo generators. While not a precise estimation of theoretical uncertainty, this comparison can provide a test of whether an observable is potentially sensitive to the different approximations of the different generators.

This sensitivity has been examined for CNN-based taggers operating on jet images in several works, and we focus here on W -tagging in a phenomenological study [66] and on quark/gluon tagging using

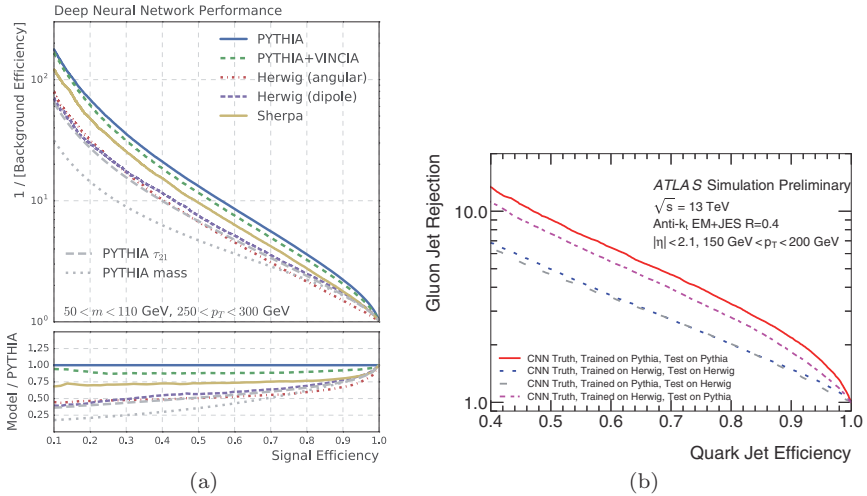


Fig. 13. (a) ROC curves for boosted W tagging for a jet image-based CNN tagger trained on Pythia generated jet images and applied to jet images from various generators are shown [66]. (b) In events with the ATLAS full detector simulation, ROC curve for quark vs. gluon tagging for jet image-based CNN tagger trained and applied on Pythia and Herwig-based jet images are shown [46].

ATLAS simulation [46]. As the CNN + jet image approaches utilize the distribution of energy throughout a jet image to discriminate, one concern is that the differences in modeling of the jet formation process by different generators may lead to large performance variations. To study this, [46] trained a CNN model on boosted W -boson jet images generated by Pythia [67, 68] and applied this trained model on samples of boosted W boson jet images generated by different Monte Carlo generators. The ROC curves of the performance can be seen in Fig. 13(a), wherein, at the same signal efficiency, reductions of background rejection of up to 50% can be seen when this tagger is applied to different generators. While such a variation is not ideal, it should be noted that similar variations were seen when a tagger of only substructure features, a binned two-dimensional signal over background likelihood ratio of the distribution of jet mass and τ_{21} , is applied for the same tagging task. Similar levels of performance variation are also seen in the ROC curves built for quark vs. gluon tagging in

ATLAS simulation with a CNN trained on Pythia jet images applied to Herwig [69] generated jet images, as seen in Fig. 13(b). Interestingly, when the test is reversed and the CNN is trained on jet images from Herwig and applied to jet images from Pythia, the tagging performance is similar to the CNN trained and applied to Pythia jet images. This suggests that the CNNs in both cases are learning similar representations of information useful for quark vs. gluon tagging, but the amount this information is expressed in the jet images varies between generators [46]. Thus while these studies show that CNNs applied to different samples may vary in performance, there may be an underlying robustness to the information learned by CNNs for jet tagging.

Beyond the potential tagging variations due to generator uncertainties, a key question when developing a jet observable of any kind is whether such an observable is theoretically sound and calculable. This is often expressed as whether the observable is infrared and collinear (IRC) safe. IRC safety for jet image-based tagging of boosted top jets with CNNs has been examined empirically in the phenomenological studies of [70]. In this work, within the context of boosted top jet tagging using a jet image-based CNN, a feature denoted Δ_{NN} is studied which explores the impact of merging soft/collinear radiation with nearby partons. Δ_{NN} is constructed as follows: (a) a CNN was trained on particle level jet images for boosted top tagging, (b) parton level jet images are generated for boosted top decays without (unmerged) and with (merged) adding the closest gluon to a top quark parton together before forming the image, (c) the difference in CNN output between unmerged and merged jet images is defined as Δ_{NN} . By examining the distribution of Δ_{NN} and its variations with features that explore soft or collinear effects, the sensitivity of the CNN tagger to IRC effects can be studied empirically. This can be seen in Fig. 14, where the 2D distribution of Δ_{NN} and the gluon relative transverse momentum, and the ΔR to the parton, are shown. As either the gluon relative momentum or the ΔR tend to zero, the Δ_{NN} distribution tends towards a sharp peak at 0, which would be indicative of the CNN being insensitive to IRC perturbations.

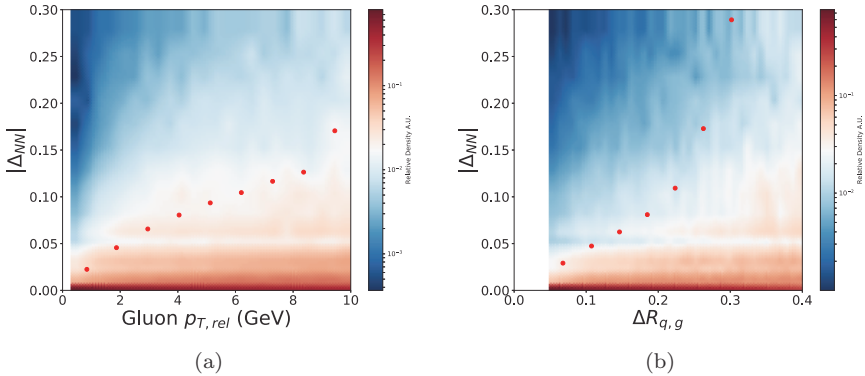


Fig. 14. Δ_{NN} vs. (a) gluon relative transverse momentum and (b) ΔR between the gluon on the nearest top decay parton. Δ_{NN} is the difference between particle jet trained CNN output applied on parton level jet images with and without merging the closest gluon with a top decay parton [70]. Red points denote the point at which 90% of events within a vertical slice of the distribution are contained.

5.4. Jet images in LHC experiments

The ultimate tests for the efficacy of jet image-based tagging approaches are that the performance observed in phenomenological studies is also observed in realistic high fidelity simulations and that their performance generalizes to real data without large systematic uncertainties. With that in mind, jet image-based tagging approaches have been examined for quark vs. gluon tagging in ATLAS simulations [46] and in CMS Open Data [60] simulated samples [61], and for boosted top quark jet tagging in CMS simulation and real data [47].

The ATLAS quark vs. gluon jet image-based CNN tagger [46] was trained using fully simulated ATLAS events [71, 72]. Multi-channel jet images were used, with one channel containing an image of the sum of measured charged particle track p_T per pixel. A second image for calorimeter measurements was examined in two forms, a jet image containing either the transverse energy measured in calorimeter towers of size $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ or a jet image containing a projection onto a fixed grid of topologically clustered calorimeter cells (topo-clusters) [13]. Translation, rotation, and normalization pre-processing was performed. A three-layer CNN with

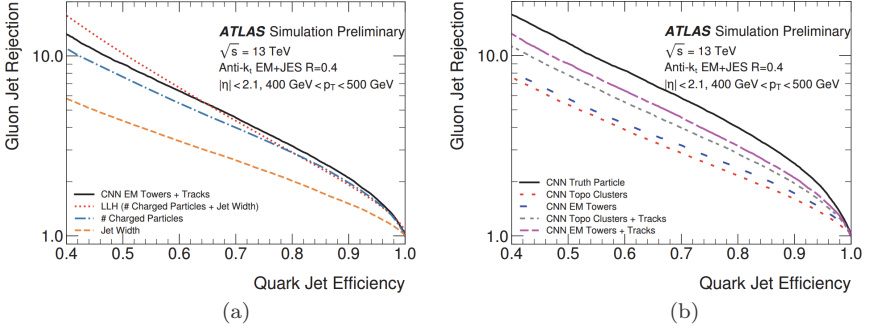


Fig. 15. ROC curves for quark jet efficiency vs. gluon jet rejection in ATLAS fully simulated datasets showing comparisons of (a) jet image-based CNN taggers against jet width and number-of-tracks discriminants, and (b) of jet image-based CNN taggers trained with different input images [46].

filter sizes of 5×5 , 5×5 , and 3×3 , respectively, and max pooling after each convolutional layer was used. As can be seen in the ROC curve in Fig. 15(a), the CNN processing the track + tower jet images outperforms other standard taggers for quark vs. gluon tagging. Interestingly, the standard tagger based on the combination of two jet substructure features (number of charged particles and the jet width) outperforms the CNN approach at low quark efficiency. This is likely due to the track image discretization that may result in multiple tracks falling in the same pixel. As track multiplicity is not stored in the images, this useful discriminating information is lost for the CNN. In Fig. 15(b), the impact on performance of utilizing different jet image channels was examined, wherein utilizing only calorimeter-based jet images provides significantly less performance than tagging using track and calorimeter images. In addition, topo-cluster-based images, which are formed by projecting the continuous topo-cluster direction estimates into a discrete grid, are seen to have lower performance than tower-based images. This is likely due to the projection onto a fixed grid for use in a CNN, as this may cause a loss of information about the spatial distribution of energy within a topo-cluster and may result in the overlap of several clusters in the same pixel. Moreover, it can be seen that the track + calorimeter image approach does not reach the performance found when a CNN

is trained on a jet image formed from truth particles (i.e. without the impact of detector smearing). It was noted in [46] that when comparing the performance of a CNN trained on only track images to a CNN trained on only charged truth particles, the observed performances were extremely similar. This similarity is driven by the excellent charged particle track resolution, and further indicates the difference between the track + calorimeter jet image-based CNN tagger and the truth particle-based CNN tagger is driven by the low resolution, and thus loss of information, of the calorimeter.

The CMS boosted top jet image-based CNN tagger [47], denoted ImageTop, was trained on fully simulated CMS events [71]. Multi-channel jet images with six channels were built using particle flow (PF) objects found within an $R = 0.8$ jet. Before pixelation, particle flow objects within the jet are pre-processed using translation, rotation, flipping, and normalization. The six channels were defined as the sum of PF candidate p_T per pixel with one channel containing all PF candidates, and one channel each for PF candidate flavor, i.e. charged, neutral, photon, electron, and muon candidates. ImageTop was based on the multi-channel DeepTop algorithm [42], and comprises four convolutional layers each using 4×4 filter sizes and max pooling after two consecutive convolutional layers, followed by four dense layers before classification. To aid the classification of top quark decays containing b -quarks, a b -tagging identification score [73] evaluated on subjets of the large jet was also fed as input to the dense layers of the tagger before classification. In addition to a baseline ImageTop, a mass decorrelated version denoted ImageTop-MD was also trained, wherein the mass decorrelation was performed by down-sampling the background quark and gluon jet samples to have the same mass distribution as the sample of boosted top jets used for training. In this way, the discriminating information from the jet mass is removed to first order.^c

^cAs the authors note, though this method is not guaranteed to remove tagger mass dependence, it was found to work sufficiently well in this case as the baseline tagger inputs were not observed to have a strong correlation to mass.

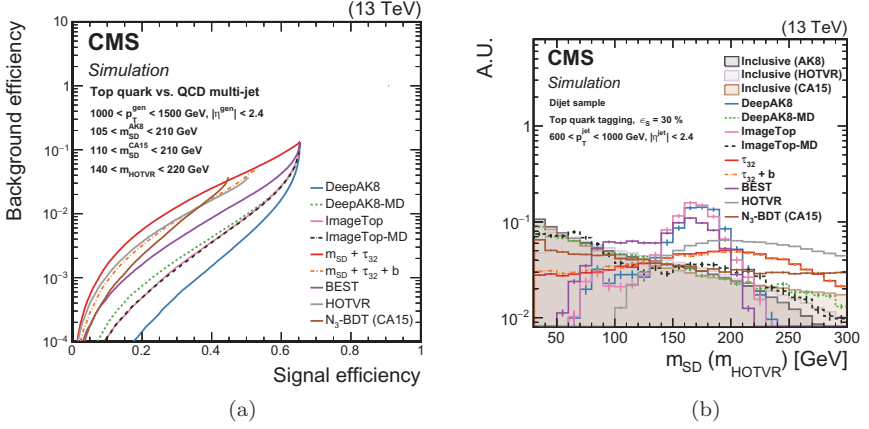


Fig. 16. Examination of the CMS ImageTop tagger [47] trained on fully simulated CMS events in (a) ROC curves of the quark/gluon jet efficiency vs. boosted top jet tagging efficiency comparing several taggers and showing the dominant performance of the deep neural net-based taggers, and (b) the impact of applying a threshold on tagger outputs to the background jet mass distribution wherein the mass decorrelated taggers show significantly less sculpting.

The ROC curve showing the performance of the ImageTop model is seen in Fig. 16(a). Several algorithms were compared to ImageTop, including several jet substructure feature-based taggers and a deep neural network, denoted DeepAK8, based on processing PF candidates. ImageTop is seen to outperform all other algorithms except DeepAK8, and generally the deep network-based taggers are found to significantly outperform other algorithms. Moreover, once mass decorrelation is included, the ImageTop-MD is found to be the highest performing mass-decorrelated model. The smaller change in performance due to mass decorrelation of ImageTop relative to other algorithms such as DeepAK8 may be due to the image preprocessing; images are both normalized and “zoomed” using a Lorentz boost determined by the jet p_T to increase uniformity of jet images over the p_T range. These steps can result in a reduction of mass information in the images and thus a reduction of the learned dependence of ImageTop on the mass. The mass spectrum for background quark and gluon jets before (in gray) and after applying a 30% signal efficiency tagging threshold for ImageTop and ImageTop-MD (in green)

can be seen in Fig. 16(b). The decorrelation method greatly helped to preserve the mass distribution and was not seen to significantly degrade performance, as seen in the ROC curves of Fig. 16(a).

As noted early, one concern with jet image-based approaches to jet tagging is their potential dependence on pileup conditions. For a fixed ImageTop tagging threshold giving an inclusive 30% top jet tagging efficiency, the variations of the top jet tagging efficiency as a function of the number of primary vertices in the event can be seen in Fig. 17. Efficiency variations for both ImageTop and ImageTop-MD were found to be small, at the level of less than 1%, across the values of number of primary vertices. A similar level of stability was observed for the background mis-identification rate. This stability draws largely from the pileup mitigation applied to the jet before creating the jet images, and this stability is not disturbed by the CNN discriminant.

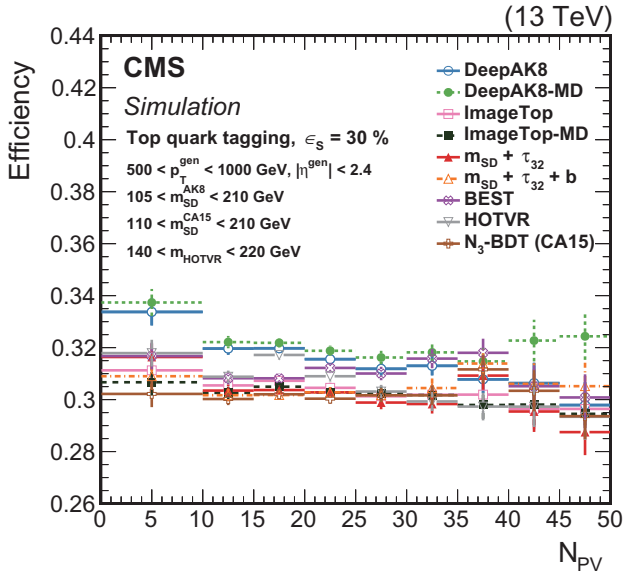


Fig. 17. Variations as a function of the number of reconstructed vertices in an event of boosted top jet tagging efficiency after applying a fixed tagger output threshold on the CMS ImageTop tagger [47], as well as several other taggers, trained on fully simulated CMS events.

While the simulation-based training of classifiers can lead to powerful discriminants, differences in feature distributions between data and simulation could cause the tagger to have differing performance between data and simulation. As such, the discriminant is typically calibrated before application in data. Calibration entails defining control samples of jets in data where the tagging efficiency and mis-identification rate can be measured in data and simulation. The efficiency of the tagger as a function of jet p_T is evaluated in data and simulation, and a p_T -dependent ratio of efficiencies known as a Scale Factor (SF) is derived. This SF can then be used to weight events such that the simulation trained tagger efficiency matches the data. The SFs for the ImageTop signal efficiency were estimated in a sample of single muon events selected to have a high purity of top-pair events in the 1-lepton decay channel, while quark and gluon background mis-identification rates were estimated in dijet samples and samples of photons recoiling off of jets. Systematic uncertainties were evaluated on the data based estimation of the tagging efficiency and propagated to SF uncertainties. These systematic uncertainties included theory uncertainties in the parton showering model, renormalization and factorization scales, parton distribution functions, as well as experimental uncertainties on the jet energy scale and resolution, p_T^{miss} unclustered energy, trigger and lepton identification, pileup modeling, and integrated luminosity, as well as statistical uncertainties of simulated samples.

The scale factors for ImageTop and ImageTop-MD for both the top tagging efficiency and the background mis-identification rate can be found in Fig. 18. The signal efficiency scale factors were largest at low momentum, showing a departure from unity of around 10%, but were significantly closer to unity in essentially all other p_T ranges. The systematic uncertainties ranged from approximately 5–10%, with the largest uncertainties at low p_T . The scale factors for the mis-identification rate tended to be larger, up to a 20% departure from unity in dijet samples but with smaller scale factors in the photon+jet samples. These calibrations indicate that while some departures from unity of the scale factors are observed, they are largely consistent with observations from other taggers. The situation is

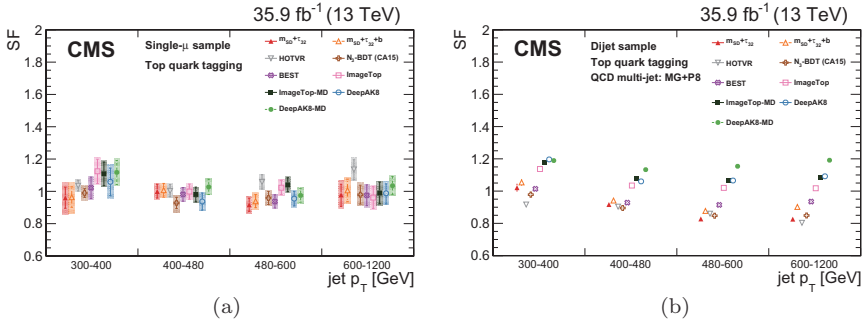


Fig. 18. Calibration scale factors as a function of jet p_T for (a) the top jet tagging efficiency in single muon events, and (b) the quark/gluon jet mistag efficiency in dijet events, for the CMS ImageTop tagger [47] trained on fully simulated CMS events and calibrated to data.

similar in terms of the scale factor uncertainties. As such, the jet image and CNN-based tagging approach can be seen to work well in data, without extremely large calibrations and uncertainties, thus indicating its viability for use in analysis.

6. Understanding Jet Image-Based Tagging

Interpretability and explainability are vital when applying ML methods to physics analysis in order to ensure that (i) reasonable and physical information is being used for discrimination rather than spurious features of the data, and (ii) when training models on simulation, models are not highly reliant on information that may be mismodeled with respect to real data. Interpretability and explainability of deep neural networks is highly challenging and is an active area of research within the ML community [74]. While a large number of techniques exist for examining CNNs, a subset of the techniques from the ML community have been applied within the study of jet images. A benefit of the computer vision approach to jet analysis is that while the data input to ML models may be high dimensional, in this case with a large number of pixels, they can be visualized on the image grid for inspection and interpretation. Thus the tools for interpreting CNN models applied to jet images tend to center on this

aspect with tools such as pixel-discriminant correlation maps, filter examination, and finding images that maximally activate neurons.

Given a jet image x with pixel values $\{x_{ij}\}$ and discriminant $c(x)$, one can examine how changes to the input may effect the discriminant prediction. Correlation maps examine Pearson correlation coefficients between each pixel and the discriminant prediction, thus probing how each input feature is correlated with increases and decreases in prediction over a sample of inputs. For a sample of N inputs, the correlation map is computed as $\rho_{ij} = \frac{1}{\sigma_{x_{ij}}\sigma_c} \sum_{k=1}^N (x_{ij}^{(k)} - \bar{x}_{ij})(c(x^{(k)}) - \bar{c})$, where $\bar{x}_{ij} = \frac{1}{N} \sum_{k=1}^N x_{ij}^{(k)}$ and $\bar{c} = \frac{1}{N} \sum_{k=1}^N c(x^{(k)})$ are the mean feature and prediction values, while $\sigma_{x_{ij}} = \frac{1}{N} \sum_{k=1}^N (x_{ij}^{(k)} - \bar{x}_{ij})^2$ and $\sigma_c = \frac{1}{N} \sum_{k=1}^N (c(x^{(k)}) - \bar{c})^2$ are the variances of the feature and prediction values.

The filters of a CNN perform local feature matching and are applied directly to the pixels of the image (or convolved image), and thus one may plot each filter as an image and examine what features each filter is targeting. As there can be a large number of filters at each CNN layer as well as a large number of channels in layers deep within a CNN, this approach tends to be easiest at the first layers of the CNN. In addition, rather than examining the filters themselves, after processing an image by a CNN model, one may examine the output of any given filter. This will produce a convolved image in which the local feature matching has been applied at each position of the image and will highlight the location of the image in which a given filter has become active. In order to highlight difference in convolved images between classes, the difference between average convolved image between two classes can highlight relative differences in the spatial location of information relevant for discrimination.

Maximally activating images or image patches correspond to applying a CNN model on a large set of images and finding the images, or image patches, that cause a given neuron to output a large activation. In the case of neurons in the fully connected layers at the end of the network, this corresponds to full images, whilst for neurons in convolutional layers this corresponds to image patches in which the neuron is most active.

6.1. Probing CNNs

In Fig. 19, the filters in the CNNs for W tagging [23] and top tagging [36] with jet images are examined. Several filters from the first convolutional layer of the CNN for W tagging are shown in the top row of Fig. 19(a), and the bottom row shows the corresponding difference between the average convolved image resulting from applying each filter. While the filters are not easy to interpret, one can see dark regions of the filters corresponding to relative locations of large energy depositions in the jet image as well as some intensity gradients that help identify regions where additional radiation may be expected. After applying the filters to sets of signal and background images and taking the difference of the average convolutions to each sample, one can explore how each filter is finding different information in signal and background-like images. The more signal-like regions are shown in red while the more background-like regions are shown in blue. The blue region at the centers identifies wider

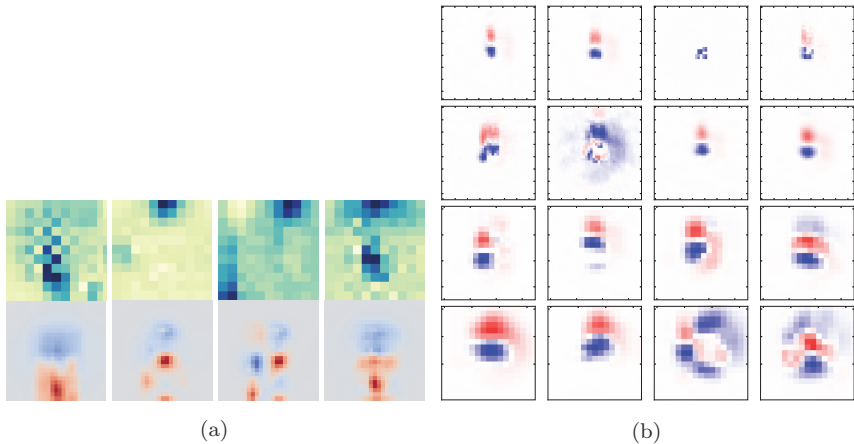


Fig. 19. (a) Filters from the first convolutional layer for boosted W tagging with a jet image-based CNN tagger [23] are shown in the top row, while the bottom row shows the average difference between signal and background-convolved images from the corresponding filter in the top row. (b) The average difference between signal and background-convolved images for several filters of the DeepTop jet image-based CNN tagger for boosted top jets [36].

energy depositions at the center of the jet image, whilst the signal-like regions at the bottom of such images identify common locations for the subleading energy deposition. There is a strong focus on identifying signal-like radiation between the leading two energy depositions. Similarly for the DeepTop model, in Fig. 19(b) one can see the convolved average image difference for several filters at each layer of the model, where the rows correspond to layer depth from top to bottom. We again see the tendency for the central region to be background-like, whilst the signal-like regions correspond to different locations of the subleading subjet and radiation between the two leading subjets. One can also see broader radiation patterns which vary depending on the location of the subleading subjet and attempt to identify likely locations of additional subjets in the image.

Figure 20 examines the average of the 500 images that lead to the highest activation for each of several neurons in the last (dense) layer of the CNN for W tagging [23]. The fraction of signal jet images in this sample is also noted, and the images are ordered left to right in terms of this signal fraction. The neuron that activates predominantly on signal jet images has a clear two prong structure and a tight core between these two prongs where radiation is expected. The neuron activating predominantly on background jet images shows a very different pattern, with a much broader central region where energy may be found and a broad ring around the central region where additional wide angle radiation may be present. These features are in agreement with the known physics of such jets.

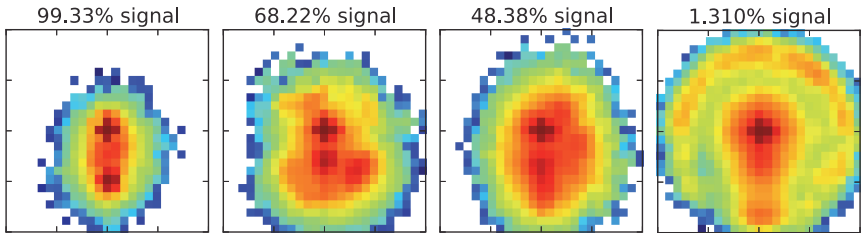


Fig. 20. The average jet image which most activates a neural in the final layer of a jet image-based CNN for boosted W tagging [23]. The fraction of signal events for each neuron is noted, thus indicating if the neuron was most activated by signal or background-like image.

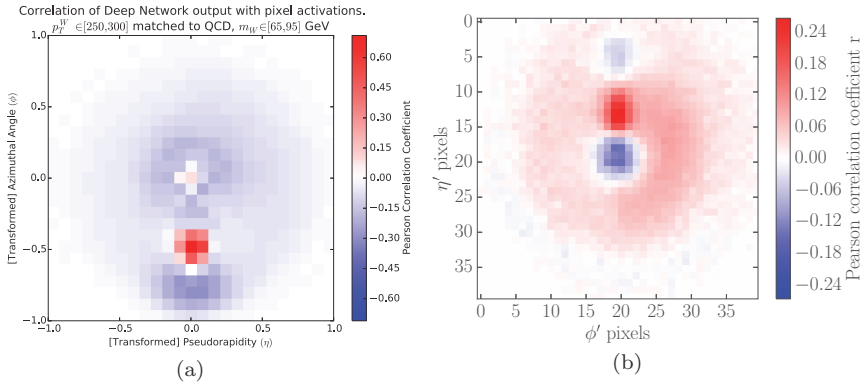


Fig. 21. Correlation images showing the Pearson linear correlation coefficient per pixel between jet image pixels and a jet image-based CNN tagger output for (a) boosted W tagging [23], and (b) boosted top tagging with DeepTop [36].

For a more global view of what the discriminant has learned, one can examine the correlation maps for the CNN W tagging [23] and the DeepTop model [36] using full preprocessing in Fig. 21. Structurally they are quite similar,^d however the regions of signal (red) and background (blue) correlation appear inverted. For W tagging, the location of the subleading subjet at the bottom of the image is a strong indicator of signal owing to the fact that W jets have a two particle decay structure which strongly restricts the relative location of the two subjets for a fixed jet p_T . This relative location is not as strict in quark/gluon jets and may vary due to additional radiation. The region around the central core of the jet is correlated with background-like images where additional radiation may be found. For top tagging, a strong energy deposition above the central leading energy deposition as well as addition energy depositions, i.e. the third expected subjet in a top quark decay, are correlated with signal-like images. This correlation pattern indicates that the discriminant relies heavily on the identification of the third subjet, as would be expected in a top quark decay.

^dThe relative location of the second subjet was rotated to be below the leading subjet in the case of W tagging and above the leading subjet for DeepTop which leads to the apparent flip in the correlation images over the horizontal axis.

7. Other Applications of Jet Images

In addition to classification tasks, the approach of using jet images and convolutional layers for processing have also been explored in several other data analysis challenges. We briefly examine some of these applications, showing how this computer vision to jet analysis can be powerful in a variety of settings.

7.1. *Jet energy regression and pileup removal*

Among the major challenges facing analyses utilizing jets at high luminosity hadron colliders is the presence of pileup, or interactions occurring in the same bunch crossing as the primary hard scattering. Pileup interactions lead to additional particles which may fall within the catchment area of a jet and thus are effectively “noise” in the estimation of jet properties. A variety of techniques have been proposed for pileup mitigation in jets [17] ranging from subtracting an average pileup energy density from a jet to techniques targeting the classification of each particle in a jet as pileup or from the hard scatter.

Within the paradigm of jet images, one approach to pileup mitigation is to predict the per pixel pileup contributions, as is done in the PUMML method [75]. In this technique, a jet can be considered as composed of four components, the charged and neutral hard scatter contributions and the charged and neutral pileup contributions. While the charged components of the hard scatter and pileup are known from charged particle tracking measurements, the neutral hard scatter and pileup components are only observed together in calorimeter measurements. PUMML performs a per pixel regression of the neutral component of the hard scatter contributions to the jet. A multi-channel jet image was used as input, with one channel for each the hard scatter and pileup charged components of the jet, and one channel for the combined neutral component. As the charged contribution measured by tracking detectors has significantly better resolution than the neutral component, a significantly smaller pixel size of $\Delta\eta \times \Delta\phi = 0.025 \times 0.025$ was used for the charged images than the $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ pixels sizes used for the calorimeter

images. Upsampling was then used to create a finer pixel image that matches the resolution of the charged component. These three channel images were then processed by a three layer CNN with a per pixel output prediction of the neutral hard scatter component of the jet.

The hard scatter neutral component prediction was combined with the known charged component to estimate jet properties and examine the efficacy of the method. In phenomenological studies using simulated dijet events produced from the decay of a hypothetical new resonance and with an average of 140 additional pileup interactions, the distributions of jet momentum and mass before and after pileup mitigation from PUMML and other methods were compared, as shown in Fig. 22. In terms of momentum prediction, comparing the pileup corrected distributions to the true distribution showed that all methods produced predictions of similar quality, though PUMML was seen to have lower per-jet reconstruction error. In terms of jet mass distribution prediction, PUMML was seen to better replicate the underlying true jet mass distribution over other techniques. While not yet applied in an experiment setting, similar ideas applied to pileup reduction for missing energy estimation have been explored on ATLAS in fully simulated events [76] and have shown promising initial results.

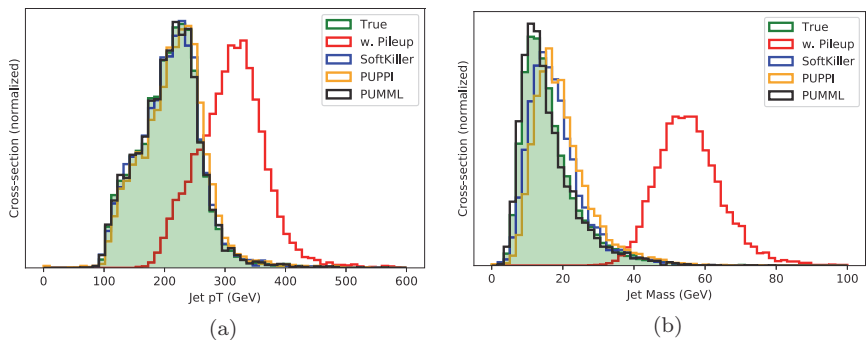


Fig. 22. The impact of pileup mitigation on (a) jet p_T , and (b) jet mass, for various mitigation techniques including the jet image-based PUMML algorithm [75].

7.2. Generative models with jet images

Among the earliest work applying deep generative models as approximations for HEP high fidelity simulators made use of jet images as the data representation [28]. The aim of this work was to learn the structure of jet images as they may appear in a calorimeter and subsequently draw sample jets from the learned generative model. As a neural generative model can be significantly faster than running a high fidelity simulator, such approaches have the potential to significantly reduce the large simulation times in HEP. In the phenomenological studies of [28], a generative adversarial network (GAN) setup was used to train a generative model to transform samples from a standard normal distribution into samples of jet images, whilst a second discriminator network was used to penalize the generative model if it could discriminate between real and generated jet images. Locally connected layers as well as convolutional layers were investigated for use in the networks. The distribution of p_T for W -boson jets and of quark/gluon jets were compared between the Pythia simulator [67, 68] and the GAN generated images, as seen in Fig. 23(a). Figure 23(b) shows a set of Pythia simulated jet images in the top row and their nearest-neighbor GAN generated jet images in the bottom row. Both the distribution of jet properties and the general

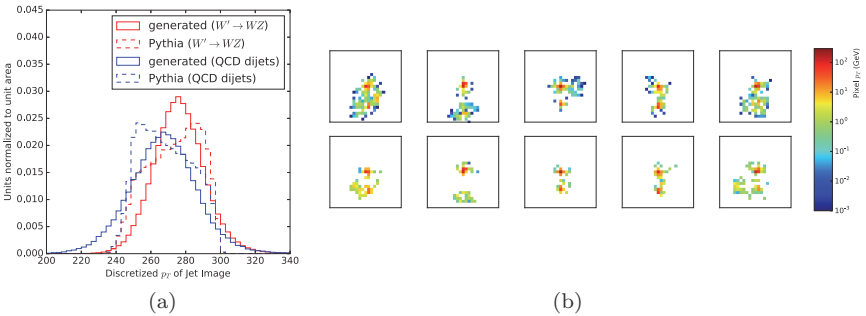


Fig. 23. (a) The jet image p_T for W -boson jet and quark/gluon jets comparing the GAN generated distribution to the Pythia simulated distribution [28]. (b) A visual comparison of Pythia simulated (top) and the nearest GAN generated (bottom) jet images.

structure of jet images were reasonably well produced by the GAN approach. While not yet reaching the fidelity of HEP simulators, this early work in HEP data generation showed the potential utility in examining fast approximation simulators from deep generative models for HEP.

7.3. *Anomaly detection*

The use of CNNs to process jet images provides a powerful scheme to learn useful representations of the information contained within a jet. In typical classification tasks, these representations are used for discriminating classes of jets. However, when searching for signs of new physics, one may not know *a priori* the properties of such a new signal but only that such a signal would have properties that deviate from known Standard Model processes. Such anomaly detection tasks are challenging due to the lack of signal knowledge and thus the inability to use standard classifiers for this task. Within the context of a search for jets produced by new particles, recent work has combined the power of CNN representation learning on jet images with autoencoder network architectures [77, 78] to search for anomalous jets [79, 80].

Autoencoder models are designed to map an input to a compressed latent representation through an “encoder”, and then decompress the latent representation back to original input via a “decoder”. Such models are trained to minimize the “reconstruction error” computed as the MSE between the original input and the autoencoder output. The reconstruction error can be used to identify inputs that are not well adapted for the compression and reconstruction scheme learned by the autoencoder. When used for anomaly detection, autoencoders are trained to compress and reconstruct one class of events. Under the assumption that this compression and reconstruction scheme would not be well adapted for inputs from classes different from the training sample, the reconstruction for inputs from new classes is expected to perform poorly and thus lead to a large reconstruction error.

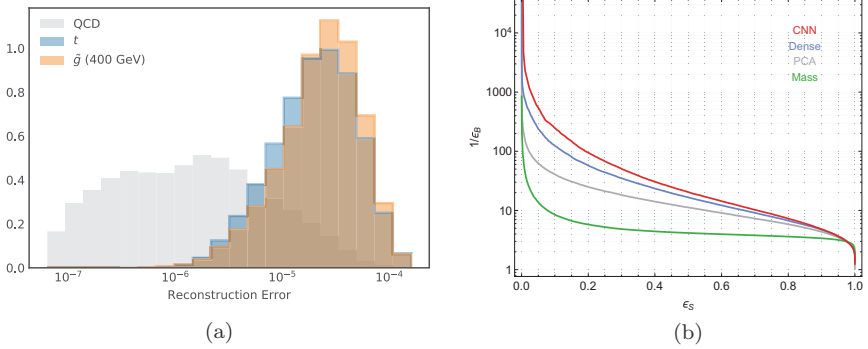


Fig. 24. (a) The distribution of autoencoder reconstruction error trained on quarks/gluon jets, showing potential top or gluino signal distributions. (b) ROC curves for identifying boosted top jets from quarks/gluon jets using autoencoders with various architectures, wherein the jet image-based CNN is shown to outperform other methods [80].

When applied to searches for anomalous jets in phenomenological studies, jet images have been examined as the data representation, and convolutional layers combined with max pooling and with upsampling have been used for the encoder and decoder, respectively. In this case, the autoencoder is trained on a background sample of standard quark and gluon jets, and the ability to identify different signal jets was examined. The reconstruction error was used directly to search for excesses of events, as seen in Fig. 24 where the signal was either a sample of top quark jets or jets from a hypothetical new gluino particle. The distribution of the reconstruction error shows a large separation from the background, denoted QCD, and the potential signal jets. The ROC curve for identifying top jets, produced by scanning a threshold on the reconstruction error, is also shown and compares the CNN-based autoencoder with dense architecture-based autoencoder applied to a flattened vector of pixel p_T 's (denoted Dense), principle components analysis, and applying a threshold only on the jet mass. The jet image+CNN architecture dominated the other methods. However, it should be noted that this domination was not seen for gluino jets.

One challenge with autoencoder approaches for anomalous jet searches is the possibility that the autoencoder reconstruction quality is dependent on the jet mass. In this case, the signal identification efficiency could be mass dependent. Moreover, if a bump hunt analysis in the jet mass spectrum is subsequently performed, such a reconstruction error correlation with mass could disturb the jet mass distribution and render the bump hunt strategy infeasible. To overcome such a challenge, an adversarial approach was investigated in [79], wherein a second network is simultaneously trained with the autoencoder to predict the jet mass from the autoencoder output whilst the autoencoder is penalized during training if the second network is successful. The resulting adversarial autoencoder performance for identifying a top jet signal can be seen in Fig. 25. With the adversary in use, the jet mass distribution was kept relatively stable even when applying a threshold on the reconstruction error which only permits a 5% background jet false positive rate. However, as seen in the ROC curve, increasing the strength λ of the adversarial

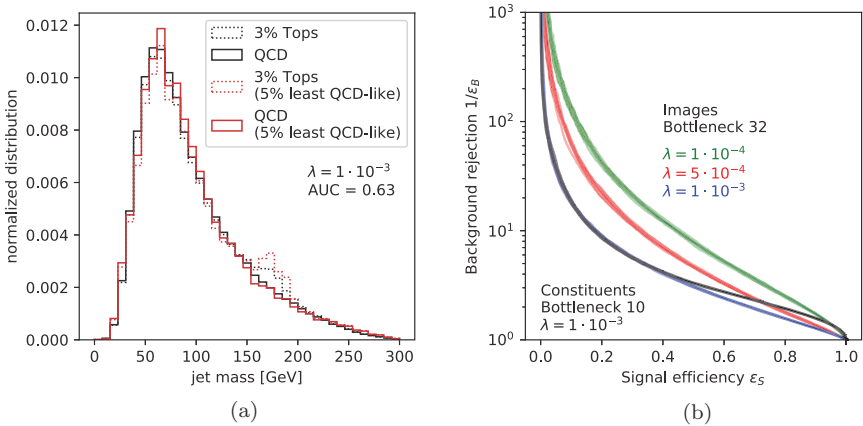


Fig. 25. (a) The jet mass distribution after applying a threshold allowing 3% or 5% quark/gluon jet background efficiency on the jet image-based adversarial autoencoder. The background is largely unsculpted and the top jet peaks can be clearly seen [79]. (b) ROC curves for quark/gluon jet rejection vs. top jet efficiency for jet image-based adversarial autoencoders with varying strength of adversarial penalty during training [79].

penalty on the autoencoder could significantly decrease the top jet signal sensitivity.

8. Conclusion

The representation of jets as images has proven highly useful for connecting the fields of high-energy physics and machine learning. Through this connection, advanced methods in deep learning and computer vision, primarily with convolutional neural network architectures, have been applied to the challenges of jet physics and have shown promising performance both in phenomenological studies and in experiments at the LHC. Jet images have seen a broad set of use cases, not only for jet classification but also for energy regression, pileup noise removal, data generation, and anomaly detection. Image-based jet tagging remains an active area of research and broad classes of state-of-the-art deep neural network architectures for computer vision are being explored within the field of high-energy physics.

While much of the work presented in this text has been in phenomenological studies using particle level simulations, there remains open question on the applicability of these methods on high-fidelity simulated data and in real experimental data. In more realistic settings, the complexity of the detector and the data-taking conditions, and the challenges of the differences between simulated and real data will be key challenges for understanding and optimizing these models. Understanding the relationships in realistic data between model accuracy and calibration error, and model complexity/structural assumptions and sensitivity to systematic uncertainties, will be important for the long-term efficacy of these image-based methods. Nonetheless, initial results from both ATLAS and CMS have shown promise, pointing towards the exciting potential for jet imaging in the future.

Acknowledgments

This work was supported by the US Department of Energy (DOE) under grant DE-AC02-76SF00515, and by the SLAC Panofsky Fellowship.

References

- [1] J. Cogan, M. Kagan, E. A. Strauss and A. Schwartzman, Jet-images: computer vision inspired techniques for jet tagging, *J. High Energy Phys.* **2015** (2015) 1.
- [2] S. Marzani, G. Soyez and M. Spannowsky, *Looking Inside Jets: An Introduction to Jet Substructure and Boosted-Object Phenomenology* (Springer, Cham, 2019).
- [3] J. Shelton, *Jet Substructure* (World Scientific, 2013).
- [4] L. Evans and P. Bryant, LHC Machine, *J. Instrum.* **3** (2008) S08001.
- [5] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, JUNIPR: a framework for unsupervised machine learning in particle physics, *Eur. Phys. J. C* **79** (2019) 102.
- [6] ATLAS Collaboration, Deep sets based Neural Networks for Impact Parameter Flavour Tagging in ATLAS, Technical Report, ATL-PHYS-PUB-2020-014, CERN, Geneva (2020); <https://cds.cern.ch/record/2718948>.
- [7] G. Louppe, K. Cho, C. Becot and K. Cranmer, QCD-aware recursive neural networks for jet physics, *J. High Energy Phys.* **2019** (2019) 57.
- [8] P. T. Komiske, E. M. Metodiev and J. Thaler, Energy flow networks: deep sets for particle jets, *J. High Energy Phys.* **2019** (2019) 121.
- [9] H. Qu and L. Gouskos, Jet tagging via particle clouds, *Phys. Rev. D* **101** (2020) 056019.
- [10] G. Kasieczka *et al.*, The machine learning landscape of top taggers, *SciPost Phys.* **7** (2019).
- [11] B. Graham and L. van der Maaten, Submanifold sparse convolutional networks (2017); arXiv:1706.01307[cs.NE].
- [12] M. Cacciari, G. P. Salam and G. Soyez, The anti- k_t jet clustering algorithm, *J. High Energy Phys.* **2008** (2008) 063.
- [13] ATLAS Collaboration, Topological cell clustering in the ATLAS calorimeters and its performance in LHC Run 1, *Eur. Phys. J. C* **77** (2017) 490.
- [14] CMS Collaboration, Particle-flow reconstruction and global event description with the CMS detector, *J. Instrum.* **12** (2017) P10003; arXiv:1706.04965 [physics.ins-det].
- [15] R. Kogler *et al.*, Jet Substructure at the Large Hadron Collider: Experimental review, *Rev. Mod. Phys.* **91** (2019) 045003; arXiv:1803.06991 [hep-ex].
- [16] A. J. Larkoski, I. Moult and B. Nachman, Jet substructure at the Large Hadron Collider: A review of recent advances in theory and machine learning, *Phys. Rep.* **841** (2020) 1; arXiv:1709.04464 [hep-ph].
- [17] G. Soyez, Pileup mitigation at the LHC: A theorist's view, *Phys. Rep.* **803** (2019) 1.
- [18] D. Krohn, J. Thaler and L.-T. Wang, Jet trimming, *J. High Energy Phys.* **2010** (2010).
- [19] D. Bertolini, P. Harris, M. Low and N. Tran, Pileup per particle identification, *J. High Energy Phys.* **2014** (2014).
- [20] Particle Data Group, Review of particle physics, *Prog. Theor. Exp. Phys.* **2020** (2020).

- [21] M. Cacciari, G. P. Salam and G. Soyez, The catchment area of jets, *J. High Energy Phys.* **2008** (2008).
- [22] J. Pearkes, W. Fedorko, A. Lister and C. Gay, Jet constituents for deep neural network based top quark tagging (2017); arXiv:1704.02124 [hep-ex].
- [23] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman and A. G. Schwartzman, Jet-images — deep learning edition, *J. High Energy Phys.* **2016** (2016) 1.
- [24] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet classification with deep convolutional neural networks, *Commun. ACM* **60** (2017) 84.
- [25] Yann LeCun, Generalization and network design strategies, Technical Report, CRG-TR-89-4, University of Toronto (1989); <http://yann.lecun.com/exdb/publis/pdf/lecun-89.pdf>.
- [26] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning* (MIT Press, 2016); <http://www.deeplearningbook.org>.
- [27] P. Baldi, K. Bauer, C. Eng, P. Sadowski and D. Whiteson, Jet substructure classification in high-energy physics with deep neural networks, *Phys. Rev. D* **93** (2016) 094034.
- [28] L. de Oliveira, M. Paganini and B. Nachman, Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis, *Comput. Softw. Big Sci.* **1** (2017) 1.
- [29] D. Scherer, A. Müller and S. Behnke, Evaluation of pooling operations in convolutional architectures for object recognition, in *Artificial Neural Networks — ICANN 2010*, eds. K. Diamantaras, W. Duch and L. S. Iliadis (Springer, Berlin, 2010).
- [30] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *Proc. 32nd Int. Conf. Int. Learning Machine*, eds. F. Bach and D. Blei, Lille, France, 07–09 Jul (2015).
- [31] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feed-forward neural networks, in *Proc. Thirteenth Int. Conf. Artificial Intelligence and Statistics*, eds. Y. W. Teh and M. Titterton, Chia Laguna Resort, Sardinia, Italy, 13–15 May (2010).
- [32] K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (2016).
- [33] L. Bottou, F. E. Curtis and J. Nocedal, Optimization methods for large-scale machine learning, *SIAM Rev.* **60** (2016) 2, 223; arXiv:1606.04838 [stat.ML].
- [34] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, in *Int. Conf. Learning Representations* (2015); arXiv:1412.6980 [cs.LG].
- [35] J. Gallicchio, J. Huth, M. Kagan, M. D. Schwartz, K. Black and B. Tweedie, Multivariate discrimination and the Higgs+ W/Z search, *J. High Energy Phys.* **2011** (2011) 69; arXiv:1010.3698 [hep-ph].
- [36] G. Kasieczka, T. Plehn, M. C. Russell and T. Schell, Deep-learning top taggers or the end of QCD?, *J. High Energy Phys.* **2017** (2017) 1.
- [37] S. Diefenbacher, H. Frost, G. Kasieczka, T. Plehn and J. M. Thompson, CapsNets continuing the convolutional quest(2019); arXiv:1906.11265 [hep-ph].

- [38] S. Bollweg, M. Haussmann, G. Kasieczka, M. Luchmann, T. Plehn and J. Thompson, Deep-learning jets with uncertainties and more, *SciPost Phys.* **8** (2020) 6.
- [39] P. T. Komiske, E. M. Metodiev and M. D. Schwartz, Deep learning in color: towards automated quark/gluon jet discrimination, *J. High Energy Phys.* **2017** (2017) 1.
- [40] Y.-T. Chien and R. Kunnawalkam Elayavalli, Probing heavy ion collisions using quark and gluon jet substructure (2018); arXiv:1803.03589 [hep-ph].
- [41] K. Fraser and M. D. Schwartz, Jet charge and machine learning, *J. High Energy Phys.* **2018** (2018) 93.
- [42] S. Macaluso and D. Shih, Pulling out all the tops with computer vision and deep learning, *J. High Energy Phys.* (2018) 121.
- [43] Y.-C. J. Chen, C.-W. Chiang, G. Cottin and D. Shih, Boosted w and z tagging with jet charge and deep learning, *Phys. Rev. D* **101** (2020) 053001.
- [44] J. Lin, M. Freytsis, I. Moulton and B. Nachman, Boosting $H \rightarrow b\bar{b}$ with machine learning, *J. High Energy Phys.* **2018** (2018) 101; arXiv:1807.10768 [hep-ph].
- [45] J. H. Kim, M. Kim, K. Kong, K. T. Matchev and M. Park, Portraying double Higgs at the Large Hadron Collider, *J. High Energy Phys.* **2019** (2019) 47; arXiv:1904.08549 [hep-ph].
- [46] ATLAS Collaboration, Quark versus gluon jet tagging using jet images with the ATLAS detector, Technical Report, ATL-PHYS-PUB-2017-017, CERN, Geneva, (2017); <https://cds.cern.ch/record/2275641>.
- [47] CMS Collaboration, Identification of heavy, energetic, hadronically decaying particles using machine-learning techniques, *J. Instrum.* **15** (2020) P06005; arXiv:2004.08262 [hep-ex].
- [48] J. Snoek, H. Larochelle and R. P. Adams, Practical bayesian optimization of machine learning algorithms, in *Proceedings of the 25th International Conference on Neural Information Processing Systems* (2012).
- [49] J. Thaler and K. Van Tilburg, Identifying boosted objects with N -subjettiness, *J. High Energy Phys.* **2011** (2011).
- [50] A. J. Larkoski, I. Moulton and D. Neill, Power counting to better jet observables, *J. High Energy Phys.* **2014** (2014).
- [51] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd edn. (Springer-Verlag, New York, NY, 2009).
- [52] M. D. Zeiler, ADADELTA: An adaptive learning rate method (2012); arXiv:1212.5701 [cs.LG].
- [53] S. Xie, R. B. Girshick, P. Dollár, Z. Tu and K. He, Aggregated residual transformations for deep neural networks, in *2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)* (2017) p. 5987.
- [54] C. Guo, G. Pleiss, Y. Sun and K. Q. Weinberger, On calibration of modern neural networks, in *Proc. 34th Int. Conf. Machine Learning*, eds. D. Precup and Y. W. Teh, Centre, Sydney, Australia, 06–11 Aug (2017).
- [55] ATLAS Collaboration, Performance of mass-decorrelated jet substructure observables for hadronic two-body decay tagging in ATLAS, Technical

- Report, ATL-PHYS-PUB-2018-014, CERN, Geneva (2018); <https://cds.cern.ch/record/2630973>.
- [56] L. Bradshaw *et al.* Mass agnostic jet taggers, *SciPost Phys.* **8** (2020) 011; arXiv:1908.08959 [hep-ph].
 - [57] G. Louppe, M. Kagan and K. Cranmer, Learning to pivot with adversarial networks, in *Proc. 31st Int. Conf. Neural Information Processing Systems* (2017).
 - [58] C. Shimmin *et al.*, Decorrelated jet substructure tagging using adversarial neural networks, *Phys. Rev. D* **96** (2017) 074034; arXiv:1703.03507 [hep-ex].
 - [59] G. Kasieczka and D. Shih, DisCo fever: Robust networks through distance correlation (2020); arXiv:2001.05310 [hep-ph].
 - [60] CERN, CERN Open Data Portal (2017); <http://opendata.cern.ch/>.
 - [61] M. Andrews, J. Alison, S. An, B. Burkle, S. Gleyzer, M. Narain, M. Paulini, B. Poczos and E. Usai, End-to-end jet classification of quarks and gluons with the CMS open data, *Nucl. Instrum. Methods Phys. Res. A* **977** (2020) 164304.
 - [62] J. Gallicchio and M. D. Schwartz, Quark and gluon jet substructure, *J. High Energy Phys.* **2013** (2013) 1.
 - [63] R. Field and R. Feynman, A parametrization of the properties of quark jets, *Nucl. Phys. B* **136** (1978) 1.
 - [64] D. Krohn, M. D. Schwartz, T. Lin and W. J. Waalewijn, Jet charge at the LHC, *Phys. Rev. Lett.* **110** (2013) 212001.
 - [65] W. J. Waalewijn, Calculating the charge of a jet, *Phys. Rev. D* **86** (2012) 094030.
 - [66] J. Barnard, E. N. Dawe, M. J. Dolan and N. Rajcic, Parton shower uncertainties in jet substructure analyses with deep neural networks, *Phys. Rev. D* **95** (2017) 014018.
 - [67] T. Sjöstrand, S. Mrenna and P. Skands, A brief introduction to PYTHIA 8.1, *Comput. Phys. Commun.* **178** (2008) 852.
 - [68] T. Sjöstrand, S. Mrenna and P. Skands, PYTHIA 6.4 physics and manual, *J. High Energy Phys.* **2006** (2006) 026.
 - [69] J. Bellm, S. Gieseke, D. Grellscheid, A. Papaefstathiou, S. Platzer, P. Richardson, C. Rohr, T. Schuh, M. H. Seymour, A. Siodmok, A. Wilcock and B. Zimmermann, Herwig++ 2.7 release note (2013); arXiv:1310.6877 [hep-ph].
 - [70] S. Choi, S. J. Lee and M. Perelstein, Infrared safety of a neural-net top tagging algorithm, *J. High Energy Phys.* **2019** (2018) 1.
 - [71] Geant4 Collaboration, Geant4 simulation toolkit, *Nucl. Instrum. Methods Phys. Res. A* **506** (2003) 250.
 - [72] ATLAS Collaboration, The ATLAS simulation infrastructure, *Eur. Phys. J. C* **70** (2010) 823.
 - [73] CMS Collaboration, Performance of b -tagging algorithms in proton–proton collisions at 13 TeV with Phase 1 CMS detector, Technical Report, CMS-DP-2018-033, CERN (2018). <https://cds.cern.ch/record/2627468>.
 - [74] U. Bhatt, A. Xiang, S. Sharma, A. Weller, A. Taly, Y. Jia, J. Ghosh, R. Puri, J. M. F. Moura and P. Eckersley, Explainable machine learning in

- deployment, in *Proc. 2020 Conf. Fairness, Accountability, and Transparency* (2020).
- [75] P. T. Komiske, E. M. Metodiev, B. Nachman and M. D. Schwartz, Pileup mitigation with machine learning (PUMML), *J. High Energy Phys.* **2017** (2017) 1.
 - [76] ATLAS Collaboration, Convolutional neural networks with event images for pileup mitigation with the ATLAS detector, Technical Report, ATL-PHYS-PUB-2019-028, CERN, Geneva (2019); <https://cds.cern.ch/record/2684070>.
 - [77] P. Baldi and K. Hornik, Neural networks and principal component analysis: Learning from examples without local minima, *Neural Netw.* **2** (1989) 53.
 - [78] P. Vincent, H. Larochelle, Y. Bengio and P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in *Proc. 25th Int. Conf. Machine Learning, ICML'08* (Association for Computing Machinery, New York, NY, 2008).
 - [79] T. Heimel, G. Kasieczka, T. Plehn and J. Thompson, QCD or what?, *SciPost Phys.* **6** (2019).
 - [80] M. Farina, Y. Nakai and D. Shih, Searching for new physics with deep autoencoders, *Phys. Rev. D* **101** (2020) 075021.

Chapter 14

Particle Identification in Neutrino Detectors

Ralitsa Sharankova and Taritree Wongjirad
Tufts University, Medford, MA 02155, USA
Taritree.Wongjirad@tufts.edu

Particle identification is a central task in the analysis of data from experiments. This is particularly true for data from Neutrino detectors where observations are made only through the products of interactions. The information available for identification varies depending on detector type. We survey various machine learning methods for particle and interaction identification. In particular, we cover applications making use of recently developed deep learning algorithms.

1. Introduction

Reliably identifying the type of particle observed in a detector is a key task for any particle physics experiment. This is no less true for experiments dedicated to studying the neutrino, which does not leave direct evidence of its path through a detector. Instead, the neutrino's flavor and kinematics must be inferred through the charged particles produced from its interactions with nuclei, which are energy dependent. Current experiments study neutrinos produced over a wide range of energies, from low energy neutrinos from nuclear reactors producing particles around 1 MeV to astrophysical neutrinos producing particles in the TeV to PeV scale. As a result, several types of detectors are in use, and the task of particle identification in these detectors will differ.

In this chapter, three broad classes of neutrino detectors will be discussed: scintillator detectors, Cherenkov ring imaging detectors,

Table 1. Detector types discussed in this chapter. This is only a subset of the detectors used in neutrino experiments.

Detector type	Signal(s)	Typical neutrino energy range	Primary source of particle ID power
scintillator	scintillation photons	0.5–10 MeV	spatial and temporal pattern of photons detected
ring-imaging cherenkov segmented	cherenkov photons ionization or scintillation photons	1 MeV–10 GeV 10 MeV–10 GeV	spatial pattern of photons detected charged particle trajectory
time-projected chamber	ionization	10 MeV–10 GeV	charged particle trajectory

and tracking detectors. These are some of the most common detectors and represent most (but definitely not all!) of detector technologies employed by future experiments. Table 1 lists the detectors discussed and the neutrino energies for which each are typically used. For each detector type, we will briefly describe the underlying physics and an example of a technique that utilizes that physics to classify particles or neutrino interactions. What the authors hope will emerge from this survey is the large range of techniques applied. However, the development of machine learning (ML) techniques for particle identification (ID) is still very much in its infancy, and there are still many possible directions to explore. Such future efforts will play important roles in the field’s effort to fully understand the properties of the neutrino.

2. Behavior of Particles in Matter

The behavior of a particle as it travels through matter is dictated largely by its charge, mass, energy, and the fundamental forces it couples to [1]. Particles with electric charge are the most visible types of particles due to electromagnetic interactions with the atoms in the detector. These interactions will produce signals that can be used to directly observe their trajectories. Electrically neutral particles like photons, neutrons, and neutrinos are observed only through the

production of charged particles via interactions with the atoms of the detector.

Charged particles when traveling through the detector transfer their energy to nearby atoms or molecules. One result of this interaction is that electrons within the atoms or molecules can be given enough energy to move into an excited orbital state, temporarily producing excited atoms or molecules. Another result is that one or more electrons are given enough energy to become unbound from the atom or molecule producing a free electron and positively charged atom, i.e. ion, or charged molecule.

The excited and ionized atoms/molecules are produced along the path of the charged particle and provide the means to infer the particle's properties. Excited atoms and molecules eventually transition back into their ground state. In the process, some transitions will produce photons that can be detected with photosensors. This process of light production is referred to as scintillation and will be discussed further in Sec. 4. There is also a special case of photon production when the charged particle exciting the molecules in the detector travels faster than the speed of light in the medium. This is called Cherenkov radiation and will be discussed further in Sec. 5. The free electrons liberated from their atoms are called ionization electrons. Ionization electrons can be collected and measured, often in a way that attempts to preserve the spatial pattern left behind by the initial charged particle trajectories. Detectors which record the spatial information of trajectories are called tracking detectors and will be discussed in Sec. 6.

Both the spatial patterns of the trajectories and the ionization created per distance traveled are used to distinguish between particles of different types. Different particles making their way through the detector will leave different patterns of trajectories due to the types of interactions available to them. The behavior of the four common charged particles detected in neutrino experiments are as follows.

Electrons have low mass and, consequently, are easily deflected when traveling through matter. When they do, they can radiate a photon. This photon can then interact with the medium by the photo electron effect, Compton scattering or photon conversion. For the

first two processes, one new electron is produced; for the latter a positron–electron pair is produced. Either way, one or two new trajectories are created and are potentially separated some distance from the original trajectory. Being electrons or positrons, they too can radiate producing more trajectories. This behavior can repeat with the end result a cascade of trajectories branching out into what is referred to as an electromagnetic (EM) “shower”.

Muons are relatively heavier and are not as easily deflected. As a result, they produce a simple linear trajectory, or “track”. However, muons are light enough such that their trajectories can be observed to wobble due to multiple Coulomb scattering.





Protons are very heavy and produce a linear trajectory with little scattering. Composed of quarks, protons can often interact with nuclei in the detector and scatter, making a sharp change in their trajectory or even producing other particles. With sufficiently high energy, there are enough additional particles that the proton induces a cascade of trajectories referred to as a hadronic shower.

Charged pions have a mass similar to a muon and produce similar “track” trajectories. Like protons they are composed of quarks and can scatter on nuclei and produce more particles. At high enough energies, pions induce hadronic showers as well. Hadronic showers can be discriminated from EM showers using their widths. Compared to electrons and photons in an EM shower, pions and protons interactions produce scattering or secondary particles at larger angles. The result is that hadronic showers are wider than electromagnetic showers.

For neutral particles, their presence is inferred through interactions which produce the above charged particles. Photons typically are observed through the production of electrons and positrons. Neutrons interacting with nuclei will produce pions and protons. Section 6 will discuss how tracking detectors make use of this trajectory information.

Finally, another key piece of information for particle discrimination is the amount of ionization left behind per unit distance traveled. This is information which can be used to distinguish between different species of particles, in particular between those that leave

Table 2. Common particles targeted for identification in neutrino experiments. This table lists the mass and the typical trajectories these particles will take within a detector for experiments where the neutrino energies are approximately 100 MeV or higher.

Particle	Mass (MeV/ c^2)	Trajectory pattern
Electron	0.511	(narrow) EM shower 
Muon	105	Long lines 
Charged pion	140	(wide) hadronic showers, lines 
Proton	938	Short lines, (wide) hadronic showers 

similar spatial patterns. For example, one would use this information to classify a track-like trajectory with no observable scattering or straggling into either a muon, a proton, or a charged pion. Electron and photons both leave behind shower patterns. However, because the photon sometimes starts a shower through photon conversion into an electron–positron pair, the amount of energy deposited at the beginning of the shower will be approximately twice as much as in a shower started by a single electron.

3. Neutrino Interactions with Matter

The properties of neutrinos must be inferred through the particles they produce via weak force interactions with nuclei in the detector.

The weak interaction is mediated either by the neutral Z^0 or charged W^\pm -bosons. Those involving the Z^0 are called neutral-current (NC) interactions; those involving the W^\pm are called charged-current (CC) interactions. When a neutrino interacts via the charged current, it produces a charged lepton with the same flavor as the neutrino: muon–neutrinos (ν_μ) will produce a muon (μ^-); electron–neutrinos (ν_e) will produce an electron (e^-); and tau–neutrinos (ν_τ) will produce a tau lepton (τ^-). Therefore, if the interaction is CC, the identification of the lepton is how the flavor of the neutrino is inferred in experiments. Detecting the absence of any charged lepton

is necessary for identifying NC interactions. Identifying the flavor of the neutrino and the NC or CC channel are often central to the targeted physics measurements.

In addition to dividing the interactions between NC and CC, interactions are grouped by the set of observable particles, referred to as the “final state”. For example, interactions can be classified by the number of protons or pions created and observed. Grouping interactions into final state is useful as each can be correlated with the ways in which a neutrino is thought to scatter with the nucleus and its constituents. A common interaction mode isolated by experiments is the charged-current quasi-elastic (CCQE) interaction which produces a charged lepton and some number of protons one can identify in the detector. The CCQE interaction is one where the neutrino scatters quasi-elastically with a single nucleon in the nucleus. Events created by a CCQE reaction are often the target of analyses because the neutrino energy can be estimated using just the charged lepton kinematics. Furthermore, having a model for a process can help with estimating the systematic uncertainties of kinematic measurements through techniques which study the variation on the predicted observation over some range of model parameter values. Describing all the different modes of neutrino-nucleus interactions is beyond the scope of this chapter. For an extensive review, please refer to Ref. [2].

4. Scintillator Detectors

Scintillator detectors count the photons emitted when charged particles travel through the detector. The detectors are composed of a large volume of liquid or solid material surrounded by photosensors. The material used for the detector contains molecules (or atoms) which, when induced into an excited electronic state, will emit one or more photons in the course of transitioning back to the ground state. The utility of these detectors derives from the dynamics of different molecular and atomic orbitals, which differ for any given material. The number of transitions that occur is proportional to the amount of energy lost by a particle as it travels through the detector. This provides the means to measure particle energy. The difference

in energy levels between transitioning orbital states dictates the frequency, i.e. color, of the photons. This allows one to choose materials that emit light with the color detected most efficiently for a given photosensor. The time it takes to transition between states for a given atom or molecule varies, ranging between “fast” transitions occurring within $O(10)$ ns or below, and “slow” transitions that are approximately $O(1)$ μ s or longer.

What gives some scintillators discriminating power between different particles is the fact that different classes of particles activate different populations of molecular and atomic transitions. This phenomenon underlies the information available to distinguish different classes of particles, which is observed as different temporal patterns in the arrival time of photons in the photosensors. The population of transitions induced is related to the density of excited molecules left behind by a particle. More massive particles deposit their energy over a smaller volume of molecules, while lighter particles deposit their energy over a larger volume. Massive particles would include protons or alphas particles, the latter of which can be emitted by decaying nuclei and is composed of two protons and two neutrons. Examples of lighter particles would be electrons or muons. For example, NE-213^a is a liquid scintillator with particle identification capabilities [3]. Neutrons, which produce a nuclear recoil when interacting with the detector, cause the medium to emit a pulse of scintillation light which occurs over approximately $O(10)$ ns. A photon interacting with the medium will produce a recoiling electron and cause the medium to emit a pulse over approximately $O(100)$ ns.

The data coming from scintillator detectors is the recorded number of photons observed over some window of time, typically between $O(10)$ ns to $O(100)$ μ s. Particle interactions in these detectors will produce a burst, or pulse, of light to be measured. From these pulses, experiments are able to extract the energy deposited into the scintillator, the position of the interaction, and particle identification.

^a “NE-213” is the name given to a commercially available cocktail of hydrocarbons with well-characterized scintillation properties. Such scintillators are given names with the initial letters deriving from the name of the manufacturer.

The energy is primarily a function of the total hits observed. The position can be derived from the spatial pattern of hits in the photosensors arrayed in the detector. Particle identification comes from the shapes of the pulses which can be different for different classes of particles due to the microscopic physics of scintillation.

4.1. *Pulse shape discrimination using CNNs*

In this subsection we describe the work in [4], which utilizes convolutional neural networks (CNNs) to perform pulse shape discrimination with a scintillator detector. The work was performed in the context of the SoLiD neutrino experiment, which aims to measure neutrino interactions from a nuclear reactor [5]. Anti-electron neutrinos are emitted from nuclear reactors with energies ranging from keVs to MeVs [6]. At these energies, the channel that is used to observe neutrinos is the inverse-beta decay (IBD) reaction where

$$\bar{\nu}_e + p \rightarrow n + e^+. \quad (1)$$

Therefore, the goal is to be able to identify the presence of a positron and neutron in the detector within some coincident time window.

The SoLiD detector uses elements consisting of a rectangular block of a type of plastic, polyvinyltoluene (PVT), with a thin plastic sheet covered in a thin film of $^6\text{LiF:ZnS(Ag)}$ affixed mechanically to one of the faces of the block. The ZnS(Ag) is a commercially available phosphor with known scintillation properties. Photosensors are coupled to the block in order to observe the light produced by both the plastic and the thin film. The plastic scintillator emits observable scintillation photons in response to the positron (electron scintillation or ES). The thin film is intended to detect the presence of neutrons in the detector through the neutron capture reaction

$$n + {}^6\text{Li} \rightarrow \alpha + {}^3_1\text{H} (4.8 \text{ MeV}). \quad (2)$$

The recoil alpha and tritium induce scintillation photons in the phosphor (nuclear scintillation or NS). Because the same photosensors will observe the light from both the PVT bulk and the phosphor screen, an algorithm is needed to identify which pulses belong to which. The discrimination between positron and neutron pulses relies on the

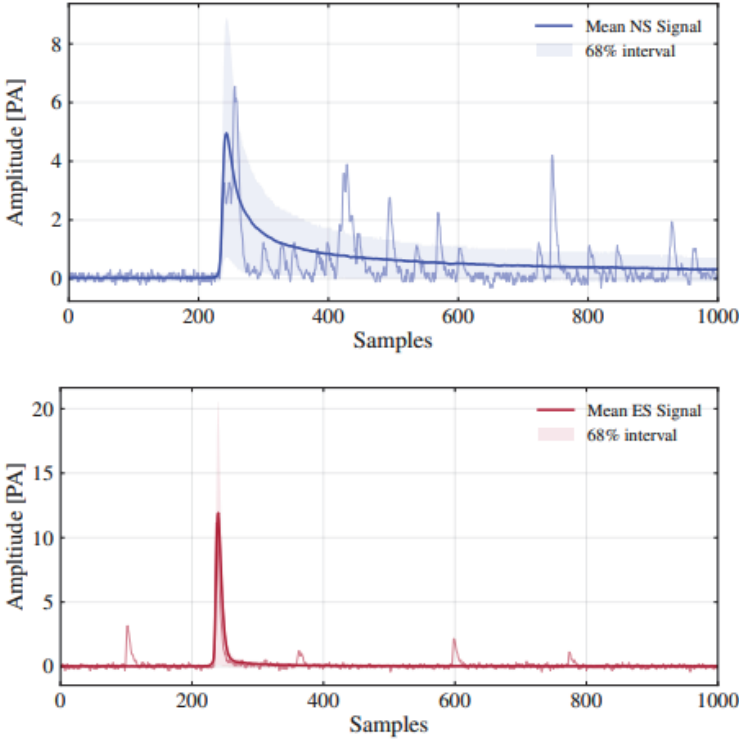


Fig. 1. Average waveforms (bold solid line) from scintillation light coming from nuclear (top) and electron (bottom) events. Signals recorded using a silicon photomultiplier (SiPM). The shaded bands represent the central 68% interval of the amplitudes. For each signal type, an example waveform is also shown (light solid line). Figure taken from [4].

expected different shape. In this case, the difference in pulse shape is driven by the use of two different scintillators, PVT for positrons and ZnS for neutrons. This is opposed to the more typical case where a single scintillator is used, but the pulse shape difference is driven by the different mixture of fast and slow scintillation components discussed above. Figure 1 shows average waveforms for nuclear scintillation (top) and electron scintillation (bottom) events.

A CNN is used to separate ES and NS pulses. The network is trained in a supervised fashion with labeled examples. Training data was obtained by taking real waveform data with a prototype setup of

a single detector element. The data was generated by recording detector responses to a AmBe radioactive source which emits both gamma rays and neutrons. Two types of photosensors observe the detector element, a photomultiplier tube (PMT) and a silicon photomultiplier (SiPM). The SiPM is the photosensor to be used in the final detector. The PMT is used in this work to provide the labels to train the network. The PMT in the prototype setup is able to collect much more light, such that the mis-labeling of waveforms is sufficiently small (see Fig. 2 in [4]). The setup was used to label SiPM waveforms, a time-series of the voltage measured from the SiPM over 1000 samples measured at intervals of 10 ns. The waveforms were labeled as either ES, coming from gamma interactions producing a recoil electron, or NS, coming from neutron capture events from the phosphor film.

The CNN architecture consisted first of two one-dimensional (1D) convolution layers each followed with a ReLU and max pooling layer. This was followed by a fully connected layer that outputs a single classification score. The 1D convolution layers used a 1D filter with size 10. The first convolution layer contained 7 channels while the second contained 14. Each max pooling layer is downsampled by a factor of two. The fully connected layer contained 64 units.

The CNN network was shown to be able to separate the two types of labeled waveforms better than two common non-ML techniques for pulse shape discrimination. The two non-ML techniques were charge integral (CI) and continuous wavelet transform (CWT). The CI technique measures the fraction of photons in a burst that arrive within some initial time window from the start of the pulse. The use of such a window distills the information contained in the pulses of light into a single number. The CWT approach is a more powerful method which uses information in both the time and frequency domain to generate features for discrimination [7]. Figure 2 compares the ROC (receiver operating characteristic) curve for three methods analyzing a separate test dataset.

Notably, this work used t-SNE [8] to visualize the 64 features in the fully connected layer in two dimensions and correlate them to the output of the CNN. This visualization is shown in Fig. 2. The work used the t-SNE to qualitatively identify the features encoded

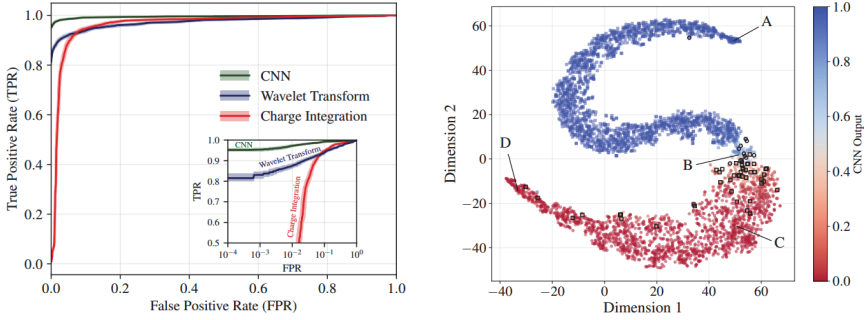


Fig. 2. (Left) ROC curves comparing classification performance between CNN (green), continuous wavelet transform (blue), and charge integration (red) algorithms. (Right) t-SNE embedding [8] of the fully-connected layer of the CNN. Each point represents a single scintillation signal. Electron scintillation signals are represented by circles, and nuclear scintillation signals are represented by squares. The color-scale represents the CNN output with red being more electron scintillation-like. Highlighted in black are mis-classified events where the NS (ES) signals that have a CNN output of less (greater) than 0.5. Points labeled with letters are examples discussed in [4] from which this figure was taken.

by the CNN and determine clusters of waveforms corresponding to event types within the ES and NS classes. For example, waveforms from muons cross the detector were identified at one extremity of the t-SNE projection (event D in Fig. 2). Events near the boundary of the ES and NS events were also identified that were likely not from either gamma or neutron events, but instead possible events consisting simply of accidental coincidences of single photon detection in the SiPM.

4.2. *Neutrino vs. background discrimination using artificial neural networks*

The work discussed here and done for the Double Chooz experiment [9] is an example of applying machine learning techniques to higher-level reconstruction quantities. In this case, an artificial neural network is used to classify events into signal and background interactions. Note that unlike the previous case, the goal is to identify neutrino interactions from events in the detector caused by other processes as opposed to individual particles. The formation of high-level

quantities, which condense the raw data into a handful of features, is representative of how physics analyses have traditionally proceeded. It will contrast in later sections with the use of representations of the data closer to the raw signals of the detector.

The Double Chooz experiment uses two liquid scintillator detectors to observe reactor anti-neutrinos from the Chooz nuclear power plant in Northern France and measure the neutrino mixing angle θ_{13} . The two detectors, called Far and Near based on their proximity to the reactor cores, are identical in structure. They consist of an Inner detector surrounded by active and passive subdetectors primarily for cosmic background rejection. The Inner detector is made up of three concentric cylindrical vessels. From the inside out those are the Neutrino Target, an acrylic vessel filled with Gadolinium-loaded liquid scintillator; the Gamma Catcher, an acrylic vessel with unloaded scintillator; and the Buffer, a stainless steel vessel filled with mineral oil and instrumented with 390 10-inch PMTs.

At 4 MeV peak energy, reactor anti-neutrinos interact inside the detector via inverse beta decay (IBD), described in the previous subsection. The products of the reaction, a positron and a neutron, are observed as a delayed coincidence of two signals. The early or prompt signal consists of the energy deposited by the positron, as well as the two gamma rays from pair annihilation. The amount of visible energy from the prompt deposits range approximately from 1 MeV to 12 MeV. The late or delayed signal is gamma rays emitted at de-excitation when the neutron is captured on a nucleus. Within the Gadolinium-loaded scintillator fluid, the majority of captures are on Gd nuclei, in which case the total energy of the delayed signal is around 8 MeV and neutron capture characteristic time is approximately $30\ \mu\text{s}$. In non-loaded scintillator captures occur primarily on hydrogen resulting in a single 2.2 MeV gamma ray released after approximately $200\ \mu\text{s}$. A major irreducible background to this signal is the so-called accidental background, which consists of accidentally coincident environmental gamma rays or a gamma ray and a spallation neutron capture. To fight accidental background, one can take advantage of the difference in high-level characteristics of the delayed coincidence between neutrinos and accidental pairs. Variables used

in Double Chooz and other reactor neutrino experiments include the physical distance between the prompt and delayed signals ΔR , the time difference ΔT , and the total visible energy of the delayed signal E_{vis_d} . Accidental background events are expected to show little to no correlation, since they are pairs of random signals. IBD events, on the other hand, show strong anticorrelation between delayed signal energy and ΔT . This comes from the overall shorter time constant for Gd captures as compared to H captures.

Earlier analyses in Double Chooz applied one-dimensional cuts on the above listed variables to reject accidental background. For the latest neutrino analysis [10] however a multivariate analysis (MVA) was employed, taking advantage of the linear correlations shown above. A multi layer Perceptron (MLP) network was chosen as the preferred MVA method and was implemented using the TMVA [11] package in ROOT [12].

Three networks were trained separately for three independent neutrino datasets: Far detector in single-detector mode (FD-I), Far detector in multi-detector mode (FD-II), and Near detector (ND). The network architectures for the three are identical, with one hidden layer and a single classifier in the output layer. Each network was trained on a sample of simulated neutrino data and a sample of accidental coincidences from data.

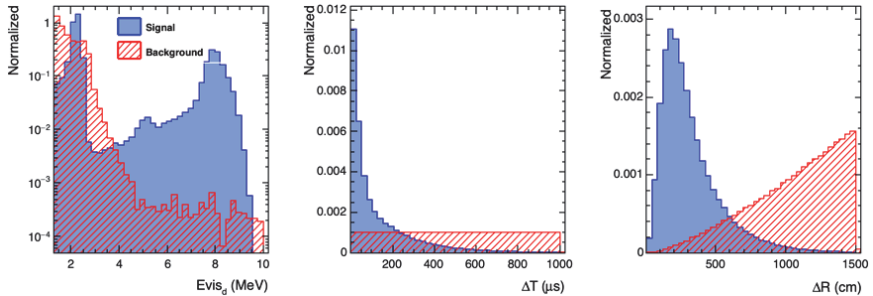


Fig. 3. High-level variables comparison between IBD signal MC (blue) and accidental BG (red). From left to right: Visible energy of the delayed signal (MeV), time difference between prompt and delayed (μs), distance between prompt and delayed (cm). Figure by authors.

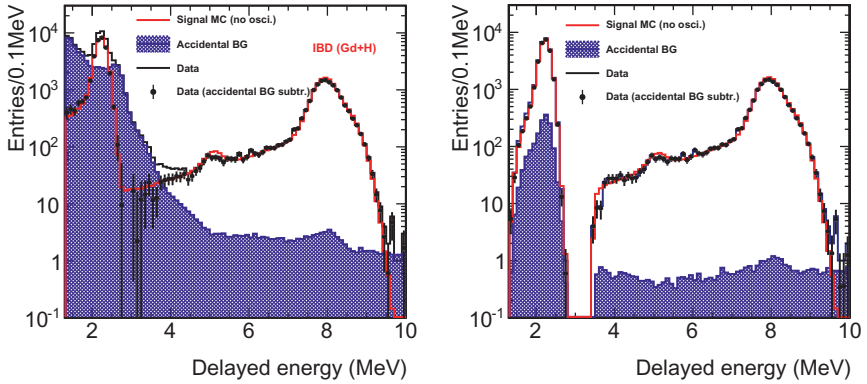


Fig. 4. Comparison of the delayed energy spectra of selected IBD candidates before (left) and after (right) applying a cut the MLP classification score. Accidental background in blue, signal MC in red solid line, neutrino candidate data in black solid line, background subtracted data in black or blue markers. FD-II data. Figure by authors.

The value of the MLP classifier used to accept events was chosen such that the signal efficiency is approximately 85% as estimated in simulated data. Requiring a classifier score above 0.85 for FD-I and FD-II sets, and above 0.86 for the ND set, one rejects approximately 97% of the accidental background events. The effect of the cut on the delayed energy is shown in Fig. 4. It can be seen that the remaining background has been sculpted to resemble IBD events.

4.3. Supervised and unsupervised classification using CNNs

The previous example distinguished events using reconstructed quantities derived from the PMT signals. The work in [13] is an example of using the PMT information more directly. It is also an example of unsupervised learning to separate event types in data with the use of labels. The work uses data from the Daya Bay detector [14]. The detector technology is similar to Double Chooz above, in that a cylindrical arrangement of PMTs monitor a volume of liquid scintillator. The PMTs are arranged only on the walls of the cylinder,

i.e. no PMTs on the end-cap, with 8 rows of 24 PMTs each for a total of 192 PMTs. This spatial arrangement is fairly straight forward to arrange in a 2D grid for use with CNNs. The work studied the use of both supervised and unsupervised classification using CNNs.

The data is prepared by forming an 8×24 image. The values of the pixels are related to the integrated charge from a single PMT waveform. The data includes low energy events, e.g. signal IBD events, and high energy cosmic muon events. To handle this large dynamic range, the value in the pixel is the natural log of the charge of each PMT. For the supervised application, the columns in the image are cyclically permuted so that the largest valued pixel is near the middle of the image, specifically the 12th column.

The sample used in the study is composed of real data events from the detector. Traditional algorithms are used to label each event as either a “muon”, “flasher”, “IBD prompt”, “IBD delay”, or “other”. “Muon” events are those that see coincident activity in muon vetos. “Flasher” events are those suspected to be due to one of the PMTs producing a flash of light. The “IBD prompt” and “IBD delayed” events are the pulses of light associated with the signal interaction. “Other”, of course, are the rest of the interactions not falling into the four categories. Note that the algorithms to classify the events rely on not just the total charge seen by a PMT after an event trigger, but also time information from the PMT waveforms and output from external detectors like a muon veto. In this study, only the spatial distribution of charge is provided.

The CNN architecture used for the supervised classification study consisted first of two 2D convolution layers each followed by a max pooling layer. This was followed by a fully connected layer that output a single classification score. For the unsupervised network, a convolutional autoencoder is used. The encoding portion of the network included sets of layers consisting of a 2D convolution operation, max pool, and ReLU activation layer. A fully connected layer takes the 2D feature maps from the convolution layers into a 10-dimensional latent space. This is then acted upon by three transpose convolution layers to up-sample the latent vector back into a 2D array the same size as the input image.

The supervised classification network performs well, correctly classifying 0.977, 0.995, 0.999, 0.974, and 0.962 of test set images with ground truth label IBD prompt, IBD delay, Muon, Flasher, and Other, respectively. The CNN outperforms other multi-variate techniques tested for comparison, k -Nearest Neighbor clustering (0.950, 0.990, 0.998, 0.891, and 0.896), and a support vector machine (0.966, 0.992, 0.998, 0.947, and 0.938). Particularly noteworthy is the analysis of the unsupervised result. Projecting the 10-dimensional feature space into 2D using t-SNE, the IBD delayed and Muon events were separated clearly. IBD prompt events were somewhat intertwined with the Other and Flasher categories. Figure 5 shows the t-SNE projection. As noted, both the supervised and unsupervised results use only the spatial pattern of the integrated PMT charge. As the two previous scintillator examples show, there is also lots more information within the time structure of light pulse seen within each PMT. Current and future efforts are focusing on bringing all of that information together.

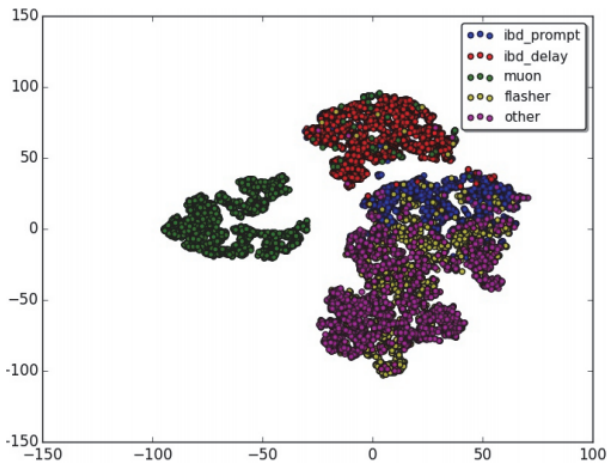


Fig. 3: t-SNE representation of features learned by the convolutional autoencoder

Fig. 5. t-SNE representation of the latent space features learned by a convolutional autoencoder trained on data from the Daya Bay experiment. Figure taken from [13].

5. Cherenkov Ring Imaging Detectors

Like the scintillator detectors described in the previous section, Cherenkov Ring Imaging Detectors observe photons. However, the photons come from Cherenkov radiation. This process occurs whenever a charged particle travels faster than the speed of light in the medium. Because the particle moves faster than light propagates in the medium the result is a wake of photon emission analogous to the shock-wave produced by a super-sonic aircraft or the wake of waves produced by a fast boat in the water. However, in this case, the photons in the Cherenkov process are emitted in the direction of the particle at an angle related to the relative speed of the particle to the speed of light in the medium. The relationship is given by

$$\cos \theta = \frac{(c/n)}{v}, \quad (3)$$

where c is the speed of light in vacuum, v is the particle's velocity in the rest frame of the detector, and n is the index of refraction of the medium. Note that $\frac{c}{n}$ gives the speed of light in the medium. For a particle with a velocity above threshold, the Cherenkov emission will produce a cone of light whose axis is in the direction of the particle velocity. Typically, a detector built to detect Cherenkov light will line photosensors along the boundary of the detector. When that cone of light hits the walls of a detector, a circular pattern can be observed with the photosensors. If the particle stopped before reaching the detector boundary, the circular pattern looks like a ring, i.e. there is a hole in the middle. If the particle crosses the boundary or gets near, the circle is filled in.

Discriminating between different particles in Cherenkov ring-image detectors relies on using the different behaviors of particles discussed in Sec. 2 and summarized in Table 2. Because electrons are light they will be deflected and induce a cascade of additional electron trajectories. This leads to many short trajectories emitting cones of light in a range of directions around the initial electron direction. The end result is a ring pattern whose outer edge is “fuzzy” compared to the ring pattern left by a heavy particle like a muon, charged pion, or proton. A muon in comparison travels in a relatively straight line

leading to a comparatively “sharp” ring pattern seen by the photo detectors. Charged pions, whose mass is similar to muons will leave a similarly sharp ring if they merely travel through the detector. However, because charged pions are made of quarks and have a sizable interaction cross-section with nuclei, they will often scatter, leaving a pattern of two or more rings in the detector. Neutral pions, which quickly decay into two gammas, can be identified by the observation of two fuzzy rings.

The different rate at which particles lose their energy is also, in principle, information one can use to assign the particle ID to a ring. Once a particle’s velocity falls below the Cherenkov threshold, photon emission stops. The angle at which photons are emitted also changes with the velocity of the particle. These effects lead to differences in the width of the ring for particles of the same kinetic energy.

5.1. *Convolutional variational auto-encoder for separating electron and gamma events*

One goal of several current and future experiments is to determine if there is CP-violation in the neutrino sector, and if so, how large? If there is, this means that neutrinos and anti-neutrinos can behave differently. Observing this fact, and seeing that the difference can be large, leads credence to scenarios that explain how the universe has come to have such an imbalance between matter and anti-matter. Even without the motivation of answering such a big question, the amount of CP-violation in neutrinos is a fundamental feature in the Standard Model that should be measured as precisely as possible.

The means to measure CP-violation will come through precision measurements of neutrino oscillations, where one neutrino can change into another. The typical experimental setup is to start with a beam almost entirely consisting of muon neutrinos and then use a detector some distance away, $O(100)$ km, in order to estimate the fraction of the beam that is now electron neutrinos. Because neutrinos cannot be observed directly, accomplishing this measurement requires identifying neutrino interactions that produce an electron. To measure CP-violation, the beam is changed into one made up of mostly anti-muon neutrinos, the fraction of anti-electron neutrinos

is measured, and the oscillation rate is compared to that measured using the original neutrino beam. Because counting neutrino interactions with electrons plays such a central role, much effort needs to go into estimating or eliminating the amount of particle interactions that can mimic electrons. This is often particle interactions made by gammas. For those gammas that Compton scatter and produce a recoiling electron, this is indistinguishable to a single electron made by a neutrino. However, gammas will often pair produce and create an electron and positron pair. Any attempt to discriminate gammas from electrons would rely on detecting the effect on the ring pattern from the additional positron.

The work in [15] studies the use of convolutional neural networks to discriminate between electron, muon, and pair production photon events. One study in this work compared a more typical, fully convolutional classification network with a classification network that used the features in the latent space of a pretrained variational autoencoder (VAE). The deterministic output of the encoder is used as the feature vector given to a 4-layer MLP. They find that for small training sample sizes, an MLP trained on the latent space features out-performed the fully convolutional classifier.

The investigation of a VAE is progress towards generative networks to produce hit patterns for particles of a given energy. Generative models which produce patterns more similar to data would improve existing likelihood-based reconstruction methods for water Cherenkov detectors [16]. Such generative models also have the potential to speed up these algorithms as producing hit pattern hypotheses is one of the costliest steps. Figure 6 from [15] provides examples of image reconstructions by a VAE.



Fig. 6. Examples of image reconstructions of simulated water Cherenkov events by a variational auto-encoder. Figure taken from [15]. The top row are images produced by the simulation. The bottom row are the corresponding reconstructions.

One last point from this work was in the preparation of the training data, which consisted entirely of simulated images. The Super-Kamiokande [17] and Hyper-Kamiokande [18] detectors, current and future water Cherenkov experiments, respectively, have a cylindrical geometry. This makes the choice of data representation an area of research. The use existing 2D techniques, a choice of projection is needed. In the work described here, the end caps are simply disregarded, which is not an acceptable choice for the actual experiments. Future work can explore different 2D projections, using 3D representations, or taking advantage of graph network methods.

5.2. *Separating neutrino-less double beta decay events from ^{10}C background with a CNN*

We discuss the work in [19] as an example application of ML for particle ID that attempts to make use of both Cherenkov and scintillation photons. The work is done in the context of current and future experiments searching for a hypothetical process, $0\nu\beta\beta$, in which a nucleus has two of its neutrons transition into two protons while emitting two electrons. The process is similar to the $2\nu\beta\beta$ process where two beta-decays occur simultaneously, emitting two electrons and two anti-electron neutrinos. The $2\nu\beta\beta$ process is possible when single beta decay is energetically forbidden while the double beta decay is allowed. For $0\nu\beta\beta$, no neutrinos are emitted. For this to occur, the neutrino must be the same as its anti-particle, i.e. the neutrino would be what is called a Majorana fermion. If true, this would make the neutrino remarkably different from the other leptons and the quarks, all of which are Dirac fermions. This has implications for the neutrino mass. If the neutrino is a different type of fermion, the way it gets its mass can differ from the other leptons, opening a large number of compelling possibilities. One is that the mass term is connected to very massive neutrino species, potentially connecting the neutrino to much higher energy scales than we currently have direct access to [20–22].

The $0\nu\beta\beta$ process, even if it exists, will have a long lifetime. Current limits set the half-life to over 10^{26} years. Any experiment searching for the process must remove backgrounds through

extremely clean detector construction and efficient background rejection. In [19], CNNs are used to separate signal $0\nu\beta\beta$ from an important source of background events coming from the decays of ^{10}C . This isotope emits gammas in the few MeV range which fall within the energy window of the $0\nu\beta\beta$ process. But the study in [19] is fairly generic in that what it tries to exploit is the very specific topology of $0\nu\beta\beta$ events.

The electrons emitted by $0\nu\beta\beta$ events will have equal energy and be traveling in opposite directions. This is due to the fact that, since there are no neutrinos emitted, the process is a two-body decay where the electron momenta are fixed to be opposed to one another. In a Cherenkov detector, what will be seen is two rings of sensor hits, on opposite sides of the detector. Most backgrounds producing events with similar total energy as the two electrons will consist of single gamma or single beta events. Coincident events will unlikely be equal and opposite in measured momentum.

The study in [19] used a CNN to perform the signal and background separation. They simulated a spherical detector with PMTs lined along the surface of the active volume and oriented towards the center of the sphere. The dimensions of the detector, the number of PMTs assumed, and parameters for the detector medium were chosen to be close to that of the KAMLAND-Zen experiment [23]. Importantly, the detector medium emits both Cherenkov radiation and scintillation light. Because there is a delay between the time a scintillator is excited and when it emits a photon, the Cherenkov radiation, which is promptly emitted, will arrive the earliest. This means that the data one wants to use is not only the spatial pattern of PMT hits but also the arrival time of photons as well. The work in [19] provides the CNN with images of hit patterns over several time windows. A visualization of this data is shown in Fig. 7.

Like all of the experiments discussed in the chapter, there are important processing choices. For spherical detectors, very common in neutrino experiments, how to treat the non-rectilinear geometry is an important consideration. Also, the data in this example is also locally-dense, but globally sparse. Sparse representation and operations can help with this issue. In this work, the spherical geometry

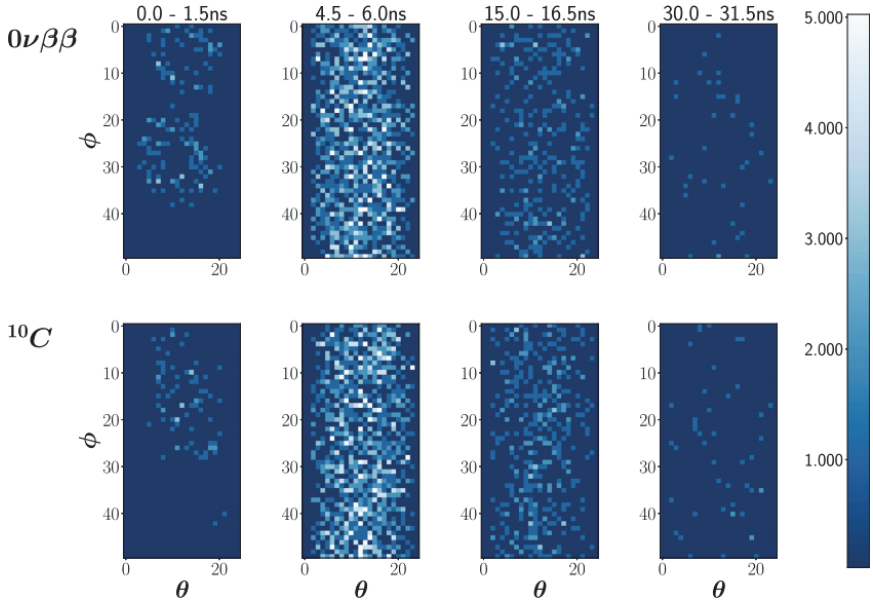


Fig. 7. Example $0\nu\beta\beta$ signal event and ^{10}C background event. In the sequence of images, each image captures the pattern of PMT hits within a specified time window for a spherical detector. Note that the early time window one can see hits due mostly to Cherenkov radiation, while the hits in the other images are mostly from scintillation photons. Image sequences like the one shown were used to train a CNN to discriminate between signal and background events. Figure taken from [19].

is handled through projection of the data into 2D. This is done by binning the hit information by the polar and azimuth angles of the PMT locations relative to the center of the detector. This choice makes it easy to represent the data. However, events with the same energy and individual particle momentum will look different when occurring throughout the detector volume. Another plausible choice would be to rely on an upstream algorithm to reconstruct the vertex and re-project the data.

With a standard detector configuration similar to the current KamLAND detector [24], the network can reject 61.6% of the ^{10}C background with 90% acceptance of the $0\nu\beta\beta$ signal. A detector with the same geometry and perfect light collection could achieve 98.2%

rejection. Performance increases to better than 99.98% for centrally located events. The overall uncertainty of the algorithm is 2.7%.

Noteworthy is the study the authors performed in trying to quantify the effect of Cherenkov light. A test sample was made for signal and background events coming from the center of the detector. One key difference in the sample was that no Cherenkov light was generated. What they find is that the performance degrades only slightly, and even increases for a smaller sized detector. This indicates that the dominant source of discriminating information is the arrival time of scintillation photons. One interpretation of this result is that further work can be done to better utilize the information from Cherenkov photons.

6. Tracking Detectors

Tracking detectors are devices that measure the trajectories of charged particles. They also measure the amount of energy lost by a particle at a given location. There are various strategies for making these measurements. In this chapter, we survey results from detectors using two common strategies. The first are “segmentation” detectors which split, or segment, the detector into sub-volumes. The collection of segments that observe energy deposited gives the information needed to reconstruct the trajectory. The second method is to use a time projection chamber (TPC). In this case, the central region of the detector is monolithic and un-instrumented. Instead, charge-sensitive electronics are placed on one side of the volume. When charged particles travel through the detector they leave behind a trail of ionization electrons which can be used to infer their trajectories. In order to measure this ionization, an electric field is applied across the active region of the detector in order to pull (drift) the ionization towards charge-sensitive electronics. The collection of sensors then records the location of charge over a series of time slices. This 2D information over time is combined with knowledge of the drift velocity and a measurement of how much time has elapsed between the moment when a particle crossed the detector and when the resulting ionization is

observed. This gives the 3D position of the observed ionization along the field direction.

The information available in tracker detectors is quite rich. This has made them an appealing target for the application of deep learning techniques, in particular CNNs which finds translationally invariant spatial patterns in image-like data. We survey such techniques here.

6.1. *Segmented detectors*

The cells of a segmented detector are constructed using materials which reflect the constraints of the experiment. For collider experiments where precision and timing are required, tracking detectors are composed of solid state silicon pixels which can provide a dense 3D array of cells providing very precise position measurements. For neutrino experiments, the detector must be large in order to compensate for the rate of neutrino interactions. As a result, the detectors are constructed using either plastic or liquid scintillator, which are relatively more cost-effective. We survey some strategies for particle ID in NOvA as an example of applications in a segmented detector.

6.1.1. *Particle identification using a CNN*

The NOvA experiment employs two detectors [25]. A near detector close to the source of neutrino beam and a far detector 810 km away. Both detectors are made up of rectangular cells of extruded, highly reflective PCV plastic filled with liquid scintillator. Each cell in the far detector is 3.9 cm wide, 6.0 cm deep, and 15.5 m long. Planes of cells are constructed by placing multiple cells along the width direction. Planes can then be arranged along the depth dimension, z , along the direction of the beam, to make a large rectangular detector. An orientation for each plane can be defined along the long dimension of the cells. The orientation of the planes are then alternated between vertical, y , and horizontal, x , as they are arranged depth wise. Both the near and far detector are constructed in this manner. The near detector mass is 300 metric-ton while the far detector is 14 metric-kilotons.

Each cell is filled with liquid scintillator which produces photons when charged particles travel through it. A wavelength-shifting fiber runs through the cell to collect the scintillation photons and guide them to the fiber ends and out onto photodetectors. As a charged particle travels through the detector, it will cause multiple cells to produce light. The 3D trajectory can be inferred from the combination of cells oriented in both horizontal and vertical positions.

Both the NOvA near and far detectors are large. As a result, a high rate of cosmic ray particles cross the detector. This is particularly true of the far detector which is near the surface. However, the experiment was designed to be able to isolate neutrino interactions using the excellent timing of the scintillator cells. Photons travel quickly and within nanoseconds of a particle crossing a cell, the photosensors register the beginning and end of a pulse of light. This fast response allows the experiment to isolate individual neutrino interactions and cosmic particles mostly through timing. Furthermore, for the neutrino interactions from the beam, the window of time during which they arrive at the detector can be calculated. Such a timing cut can be used to isolate pure samples of neutrino interactions.

The data from the segmentation cells are natural 2D images. Recall that cells are oriented in two directions. By recording the amount of light seen in each cell and arranging data from the cells in either the horizontal or vertical orientations one forms a set of two 2D images representing the XZ and YZ projections of the event. Figure 8 provides a schematic of the NOvA detectors along with illustrations of the XY and YZ views.

One of the primary goals of the NOvA experiment is to measure the rate at which muon neutrinos created by the Fermilab accelerator change into electron or tau neutrinos by the time they arrive at the far detector. This change from one type of neutrino to another is possible through the phenomenon of neutrino oscillation. A description of this is outside the scope of this chapter. See [27–29] for reviews of recent experimental results on neutrino oscillations at the time of writing. Therefore, in the end, the task on hand is to count the number of muon neutrinos and electron neutrinos at the near and far detector.

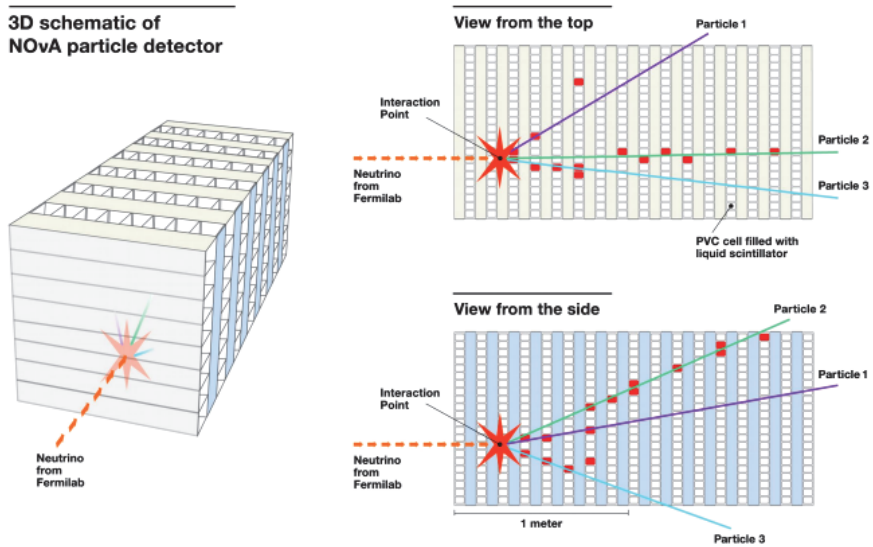


Fig. 8. Schematic of the NOvA detectors. (Left) An illustration of the 3D detector. Illustrations of the detector from the top (top right) and side (bottom right) are shown. These views help to show how images of neutrino interactions in the NOvA detector are formed. For each view, one can form images giving the XZ projection (top right) and YZ projection (bottom right) by ignoring the cells that are oriented vertically with respect to the view. Such interaction images are used in a CNN for classifying neutrino interactions in NOvA. Figure taken from [26].

NOvA pioneered the application of CNNs to neutrino experiments [26]. The first application targeted the identification of muon neutrino interactions from electron neutrino interactions. This was not particle ID for individual particles but rather for the interaction on the whole. However, as described in Sec. 3, the flavor of the neutrino is only known if the partner lepton is produced via a charge-current interaction. This implies that the network must learn to reliably detect the presence of either a muon or electron within the entire interaction image. For more details on this, see Chapter 13.

The reason that whole interaction classification was the natural first application and not individual particles is the need in tracker detectors to cluster the locations of energy deposits in two individual particle trajectories. Because of the scintillation pulses last for

approximately less than $O(1) \mu\text{s}$, the detector has the ability to isolate individual interactions through timing. However, the time resolution is not fast enough to separate the individual charged particle trajectories. Because of the need to cluster images, particle identification (PID) algorithm using supervised-learning techniques is likely affected by the quality of the clustering algorithm. The clustering algorithm acts as a pre-processing step which will produce mistakes that the PID must learn to account for. If the clustering is poor, this potentially introduces enough noise in the labels to limit the performance of the algorithm.

The work in [30] approaches this clustering issue by simultaneously providing a convolutional neural network images of individual trajectories in addition to the full, un-clustered interaction image. The authors refer to this as “context-enriched particle identification”. The principle is that the context of the particle within the interaction provides information to make a more accurate classification. For example, electrons and photons both produce EM shower patterns in the detector. Inspecting the showers alone, the dE/dx in the very beginning portion of the shower is the primary handle to resolve between an electron or photon induced shower. With the whole interaction in view, the identification of a second shower increases the likelihood that both showers were produced by photons from the decay of a neutral pion. This is an example of information that the network could use.

The network architecture used consisted of four-towers with the same sequence of operations. Four towers were used because a total of four images per particle are provided to the network. This consisted of two sets of a particle-only and full-interaction image pair, with one set for each of the two views of the detector, XZ and YZ. Figure 9 shows examples of images from the NOvA detector along with examples of the trajectory clusters. The first set of layers apply a sequence of 2D convolution layers followed by one GoogLeNet inception module. The output of the four towers are then concatenated and passed into one inception module. A final convolution layer produces scores for five particle types: electrons, photons, muons, charged pions, and protons.

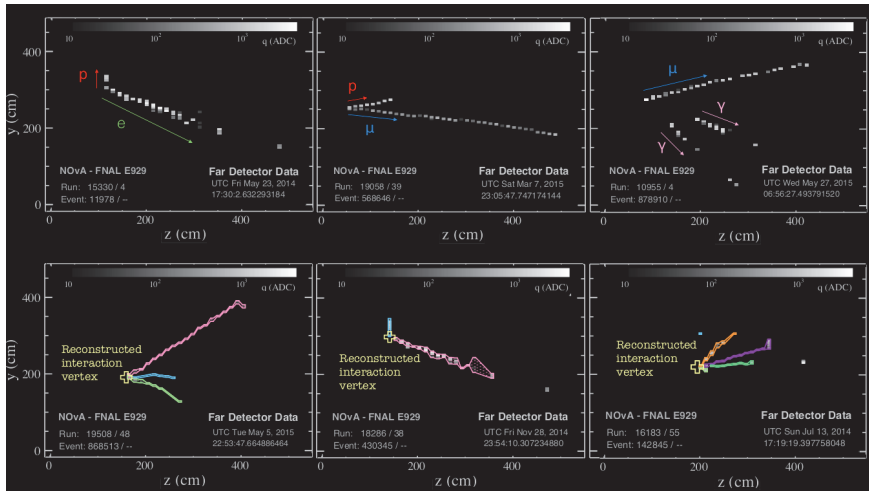


Fig. 9. Examples of neutrino interaction images from the NOvA detector. The top row shows examples of full interaction images passed into a “context-enriched” particle identification network. They also show examples of muons, protons, electrons, and photons. The annotations are to show possible particle type assignments and not part of the context given to the network. The network, in addition, to full interaction images are provided images where only the trajectory in question is shown. The bottom row shows more interaction images with individual particle clusters outlined. The reconstructed neutrino vertex is indicated by a cross. The pixels inside the contours are those used to form individual particle images, one for each cluster. Figure taken from [30].

The algorithm that forms the clusters is based on fuzzy K-means [31]. It is applied over the set of angles with respect to the reconstructed neutrino vertex of cells with energy observed over threshold. First clusters are formed in each of the views. Next, these clusters are matched between the views based on their consistency in energy and location. According to studies using simulated data with available truth information, the algorithm produces clusters which contain nearly all of the cells into which a given particle deposited energy. The purity of a cluster, however, is lower as each cell tends to contain energy deposited by overlapping particles.

The set of clusters used to form the training data are a subset of all clusters formed. First, clusters shorter than 5 m are used. Longer clusters are very likely muons and so there is not much need for the

classifier for these clusters. Next, a cut on purity is applied. True muon, electron, and photon clusters are required to have purity of 0.5. True pion and proton clusters are required to have a purity of 0.35. This was done to isolate examples with minimal overlap. The training set consisted of 2.95 million clusters.

The network correctly classifies 93%, 75%, 93%, 65%, and 81% of true electrons, photons, muons, pions, and protons, respectively. Of those labeled electron, photon, muon, pion, and proton the fraction correctly labeled is 90%, 75%, 87%, 54%, and 89%, respectively. The largest mis-identifications come between electrons–photons and protons–pions as might be expected. Electrons and photons are confused due to their similar topology. Protons and pions are confused due to their typically short trajectory length. When compared to a network without context information, the increase in classification accuracy is seen in photons and pions. The non-context network was implemented with only two towers, one for each single-particle image, where each tower contained the same number of layers and features per layer. The context-enriched network is 11% more accurate for in correctly labeling photons and pions. True electrons mis-identified as photons reduces by 5% when using the context-enriched network. True photons mis-identified as electrons reduced by 3%. Pions mis-takenly classified as photons is reduced by 8%.

6.2. *Time projection chambers*

Time projection chambers differ mostly in the target material used and the strategy for reading out the ionization from particle trajectories. TPCs were first developed using gas. Electron drift velocities range from $\text{mm}/\mu\text{s}$ to $\text{cm}/\mu\text{s}$. For liquid detectors, drift velocities are of the order of $1 \text{ mm}/\mu\text{s}$. The drift velocity determines the time it takes to draw ionization electrons to the instrumentation. Within that time other particle trajectories can cross the volume, especially if the detector is near the surface where the number of cosmic rays detected in time with particle or interaction of interest can begin to be large. For machine learning techniques, this confronts one with the need to isolate the particle of interest before being able to evaluate the particle. This leads to potential background in the test and

training sample. For algorithms like CNNs, the isolation of the particle requires what amounts to image pre-processing, whose effects on training and testing one needs to consider.

6.2.1. *Signal and background separation for neutrino-less double beta decay*

The Neutrino Experiment with a Xenon TPC, or NEXT [32], is dedicated to searching for neutrino-less double beta decay ($0\nu\beta\beta$). The NEXT detector employs a TPC filled with high-pressure xenon gas. The operation of the detector is similar to the other TPCs previously discussed. However, one way in which it differs is in the way it records to the 2D position of ionization clusters near the anode of the drift region. Instead of charge-sensitive electronics recording the induced current, PMTs sit outside the drift region and face into it. The drift region near the PMTs is further segmented by a wire mesh into an additional amplification region. Here an electric field, larger than the one used to drift the electrons, is used to accelerate the ionization electrons. The goal is to impart enough energy to the ionization electrons so that they begin to multiply through the liberation of additional atomic electrons and also induce the emission of scintillation photons by the xenon. The end result is that a charged particle leaves behind a trajectory of ionization that is seen as a series of 2D patterns of light flashes occurring over time. This information can be used to reconstruct the 3D trajectory.

The goal of the NEXT experiment is to search for the set of trajectories coming from $0\nu\beta\beta$. The signature for this process, initially described in Sec. 5.2, is two electrons emitted with the same energy, back-to-back. The total energy of the electrons coming from the decay of a Xenon nucleus is expected to be 2.46 MeV [33]. The dominant backgrounds to observing $0\nu\beta\beta$ will be from nuclei which emit beta or gamma particles around this energy. Examples are gamma rays of energies 2.447 MeV and 2.614 MeV, emitted by daughters of ^{214}Bi and ^{208}Tl , respectively.

The key topological handle the detector will have to separate signal from background is the fact that the signal decay consists of two

electrons. The background events may consist of (1) a single electron or positron from beta decays, (2) a single electron from a gamma Compton scattering on an atomic electron, or (3) an electron and positron pair from a gamma converted through the pair-production process. The fact that the signal will have two particles means that one can look for two electrons stopping in the detector and the resulting two Bragg peaks. Electrons coming to a stop produce a large amount of ionization at the end of the trajectory. The presence of two such balls at the end of a contiguous cluster of ionization is the feature that one targets in trying to select signal events. Figure 10 provides an example event display of two electron trajectories from $0\nu\beta\beta$.

The reason single electron background trajectories can mimic the two-end-blob topology is due to the large fluctuations of ionization that can occur while charged particles travel through matter. The distribution of ionization produced per unit distance traveled is peaked, but has a long tail. For single electron background events, if a fluctuation in the ionization occurs sufficiently close to the beginning of

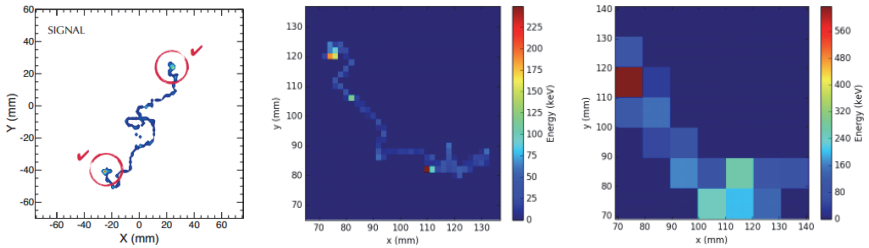


Fig. 10. (Left) Simulation of a signal ($0\nu\beta\beta$) event in xenon gas at 15 bar. The color corresponds to energy deposition in the gas, red representing higher density of energy deposition and blue representing lower density. The signal consist of two electrons emitted from a common vertex (figure from [34]). The trajectory is shown at a resolution higher than what the NEXT detector can measure. (Middle) XY projection of signal trajectory voxelized with $2 \times 2 \times 2 \text{ mm}^3$ voxels. (Right) Same signal trajectory as the middle figure but binned with $10 \times 10 \times 5 \text{ mm}^3$ voxels. The middle and right image, taken from a figure in [33], are examples of the data passed into a CNN.

the electron trajectory, this can produce an ionization cluster that looks like it has two end-blobs.

A study was performed investigating the ability of a CNN to distinguish signal from background events [35]. The data from the detector was presented to the CNN as a set of three 2D images made by taking the projections of the 3D ionization data. Two image resolutions were studied, based on the size of 3D voxels used to bin the ionization data before projection. The first is $10 \times 10 \times 5 \text{ cm}^3$ and the second is $2 \times 2 \times 2 \text{ cm}^3$. The first is a resolution close to that demonstrated by current NEXT prototypes. The latter is a resolution closer to the expected capability of the detector. The resolution is expected to play a large role in being able to reject background events as it directly impacts the ability to determine that a large amount of ionization away from the trajectory end.

The current limit on the half-life $0\nu\beta\beta$ is $O(10^{26})$ years. If the process can occur, the number of events in the detector will be small compared to other backgrounds, even after the impressive efforts in background mitigation realized through meticulous choices of materials and assembly techniques. This emphasizes the need to use some pre-processing algorithms on the data to remove events that are very-likely backgrounds. Only the more difficult cases are given to the CNN, which makes a decision as to the class of the event as a last step. The pre-processing steps applied in the study involved (1) a cut on the total energy to be between 2.4 and 2.5 MeV, (2) a cut on the distance away from the edge of the TPC boundary, and (3) the presence of a dis-joint ionization cluster. The latter suppresses gamma events where an initial electron trajectory is created by Compton scattering and then interacts again near-by either via another Compton scatter or via pair production.

The GoogLeNet [36] architecture was used in the studies. Both the signal and background examples are first passed through the pre-processing steps above. As a comparison, an expert-made algorithm was also applied that looks for the end of the trajectories and sums the ionization at those locations. Two-blob events were then identified in large part by the endpoint ionization. The CNN slightly out-performs the expert-algorithm. For the coarse-grained resolution,

the background acceptance drops from 11.0% to 9.4%. For the finer-grained resolution, the background acceptance drops from 7.6% to 4.7%. As expected, the use of higher-resolution images improves the performance.

A note-worthy study in this work is the attempt to relate performance of the CNN to the microscopic physics of the electron trajectory. The tools used to produce simulated electron trajectories can be modified to include or exclude particular physics effects. The work presented here, toggled through a series of simulation configurations which was shown to modulate the performance of the CNN from near perfect to the observed performance when including all the expected physics. For example, the expected fluctuation in the ionization amount per distance traveled can be turned off and replaced by a constant amount of ionization. As expected, the ionization fluctuations were one of the largest contributors to mistakes. When possible, the intention in correlating the performance to these physical effects is to increase the credibility that the networks are using information from physics processes related to real differences in the signal and background events.

6.2.2. *Particle ID in a liquid argon TPC*

The MicroBooNE detector is a TPC where liquid argon is used as the target medium [37]. Cryogenic liquid noble gases are very inert and can be purified to the levels of several parts per billion of contaminants to liquid argon molecules. It has been shown that liquid argon with such purity can accommodate the drifting of ionization electrons over meter-long distances [38]. This property along with the fact that liquid argon is a relatively inexpensive material, allows liquid argon TPCs (LArTPCs) to scale up to large sizes. The use of a cryogenic liquid also means that the detector medium is several times denser than gas and can therefore proportionally increase the rate of neutrino interactions that occur. This has made LArTPCs the choice for several current and future neutrino experiments.

The downside to LArTPCs is that the time to capture an event is relatively long for particle physics detectors. For example, the

dimensions of the MicroBooNE TPC are $2.56 \times 2.33 \times 10.36$ m. The drift field is oriented along the dimension whose length is 2.56 m. With an applied potential of 75 kilovolts, the drift velocity of ionization electrons is 1.098 ± 0.0044 mm/ μ s [39], making the time to drift across the entire detector 2.3 ms. This time window is long enough to cause challenges. Assuming that it is known that an event of interest occurs somewhere in the detector, to ensure the ionization electrons are measured, one must collect data over the full drift windows. For MicroBooNE, this means recording data around the known time a neutrino beam will pass through the detector. This beam window is 1.6μ s wide and much smaller than the drift window. As a result other particles can enter the detector within the drift window and will complicate the task of identifying specific types of particles or neutrino interaction types. The effect is that particle or interaction ID is dependent on the performance of algorithms dedicated to clustering locations of ionization. For MicroBooNE, which operates near the surface, the complication is quite large as a neutrino interaction, even if one occurs within the drift window, is accompanied by roughly 10–15 interactions coming from cosmic rays. This point is emphasized because the examples of particle or interaction ID that follow either assumes clustering has already occurred or even produces information related to particle ID in order to assist clustering.

Single Particle Classification: In this work [40], particle ID in the MicroBooNE LArTPC was studied assuming perfect clustering. The training and test data consisted of images cropped around the simulated trajectories of single particles. Five different particle species were used: μ^- , p , π^+ , e^- , and γ . The particles were generated uniformly over the detector and over a range of particle kinetic energies. The direction of the particles were generated isotropically. Crops of 576×576 pixel images were made by partitioning the detector in 3D regions. The effective result is that the particle can be located anywhere in the cropped image. Figure 11 from [40] shows an example image from one of the different particle types.

We focus on the CNN performance on a particular pair of particles, electrons and photons, which is an important for LArTPC experiments studying neutrino oscillations. Like other experiments

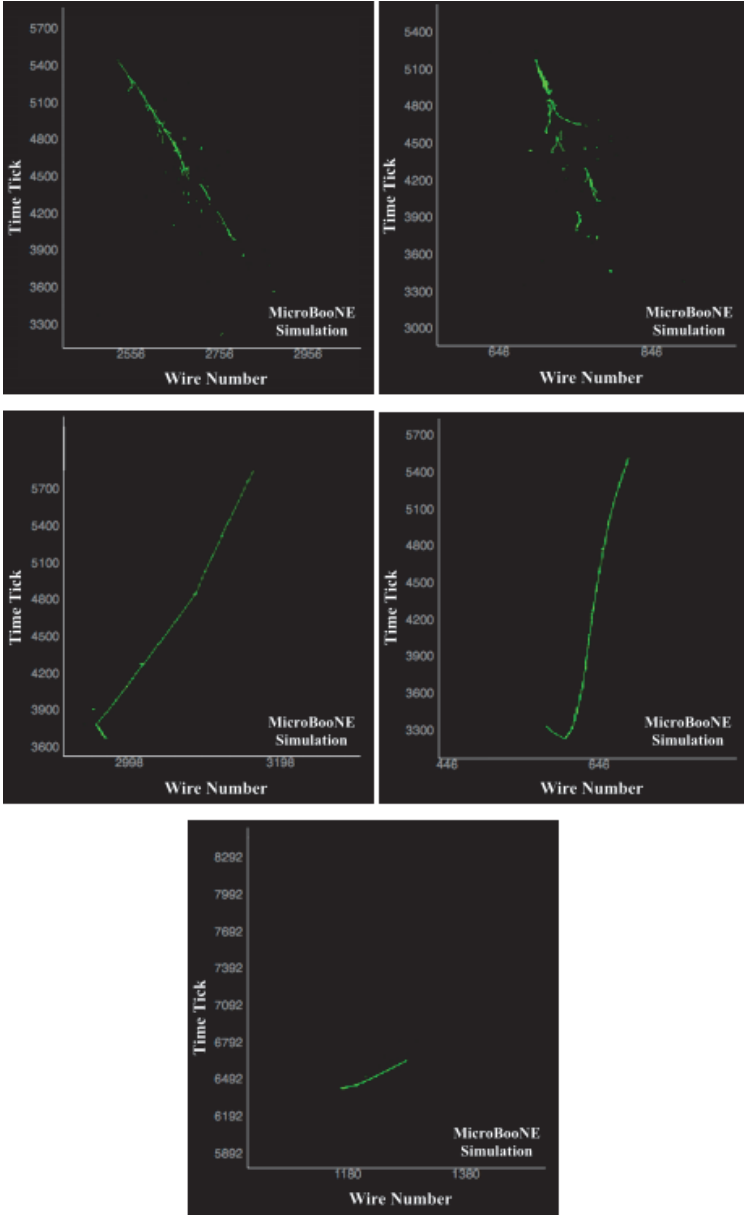


Fig. 11. Examples of the five particle types in the MicroBooNE detector: (top left) electron, (top right) gamma, (middle left) π^- , (middle right) μ^- , and (bottom) proton. Figure taken from [40].

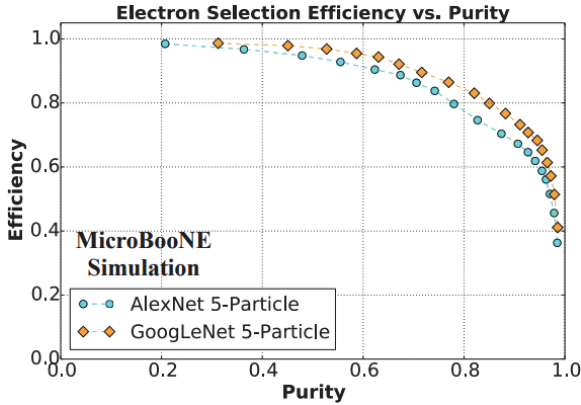


Fig. 12. Electron selection efficiency vs. photon rejection curve for two different CNN architectures, GoogLeNet and AlexNet, trained for 5-particle classification. Figure taken from [40].

described in this chapter, interactions that produce photons are one of the largest backgrounds in MicroBooNE in measurements of electron neutrino events. The ability to separate electrons from photons is shown in Fig. 12. The furthest extent in the electron efficiency vs. purity achieves an efficiency of $83.0 \pm 0.7\%$ with a purity of 82.0%. There is minimal confusion between electrons and photons, which produce shower-like trajectories, with particle types that make track-like trajectories, muons, pions, and protons. Note that the efficiency values shown are for test images generated over the same momentum ranges as the training set, i.e. uniformly between 100 MeV to 1 GeV, and not over a distribution of electrons and photons that the MicroBooNE experiment expects to see.

Track-Shower Semantic Segmentation: The network described above, assumed that particle clustering has already been perfectly applied. In contrast, in [41] the authors targeted network outputs that could be used to seed potential particle clustering algorithms. The goal of the network was to label pixels in the LArTPC images as either deriving from particles that produce a track-like ionization pattern, i.e. μ , p , π^\pm , or deriving from particles that produce a shower-like ionization pattern, i.e. e^\pm , γ . The network was based

on fully convolutional semantic segmentation network. The network used a U-Net architecture [42] with residual convolution layers.

The data used for training was generated with simulations where a random number of particles, uniformly distributed from one to four, were emitted from a common vertex. For 80% of the sample, one of the particles had to be an electron or a muon. The maximum multiplicity for leptons and protons had to be three and for photons and charged pions — two. The directions of the particles were distributed isotropically, while the kinetic energy of the particles were generated uniformly over a range between 50 to 1000 MeV. For the other 20% of the sample, the presence of one electron or muon is not required and the maximum multiplicity for any particle is two. The goal for this portion of the sample was to provide low energy examples, and so the ranges for the randomly assigned momenta were 30–100 MeV/ c for electrons and photons, 85–175 MeV/ c for muons, 95–195 MeV/ c for charged pions, and 300–450 MeV/ c for protons. The individual choices for multiplicities were random. But the point of generating images in this fashion was to avoid the use of a neutrino interaction generator which would build in assumptions on the multiplicity and distributions of momentum. The multi-particle generator was an attempt to give a wider range of possible final state configurations, including even non-physical ones. To quantify the effect of this choice, the accuracy is calculated for specific neutrino interaction samples with a range of particle energies. Table 3 compares the incorrect pixel fraction per event for the test set, made with the same particle kinematics as the training sample, along with different samples of interest to MicroBooNE analyses.

One aspect that is noteworthy in the training of the network was the use of pixel-wise weights in the loss function. The goal was to use the pixel labels upstream of not only clustering, but also candidate neutrino vertex formation. This meant that not all pixels were of the same importance. Instead, it was vital that pixels at the boundary between differ classes of particles would be considered important, with pixels near the neutrino interaction vertex most important of all. This motivated the weighting of pixels based not only to account for the frequency at which a given label appears, but also on their

Table 3. Summary of the performance of a CNN tasked with assigning pixels as either track-like or shower-like. The metric shown is the mean and 90%-percentile incorrect pixel fraction (ICPF) per image. Also shown in the last two columns are the mean ICPF for true shower and track pixels. The test set sample was made in the same fashion as the training images by generating particles from a common start point without the use of a neutrino interaction generator. The metrics for the test set are compared to samples that incorporate estimates of the ν_e and ν_μ flux as seen by MicroBooNE and interaction cross-sections as modeled by the GENIE generator (second and third row). The last three rows of the table show the metrics for specific final state that are the target of physics searches. These are 1 electron and 1 proton (1elp) events, low energy 1 electron and 1 proton (1elp-LE) events, and low energy 1 muon and 1 proton (1 μ 1p-LE) events. Figure taken from [41].

Sample	ICPF mean	ICPF 90%	Shower	Track
Test	1.9	4.6	4.1	2.6
ν_e	6.0	13.8	7.6	13.8
ν_μ	3.9	4.5	14.2	4.3
1elp	2.2	5.7	2.8	4.0
1 μ 1p-LE	2.3	2.2	6.2	2.4
1elp-LE	3.9	11.5	3.8	8.0

importance to downstream reconstruction stages. Figure 13 provides an example multi-particle image used in the training sample. The different classes of pixel-wise weights are also shown.

In this work, there are several studies comparing the behavior of the network on real and simulated images. Two event samples were prepared to do such studies. The first is a sample of images containing a stopping muon that decays into a Michel electron. The second sample was composed of ν_μ -CC interactions where a π^0 is identified. This sample was selected using different, non-ML reconstruction algorithms. Data samples, both from real data and from simulated

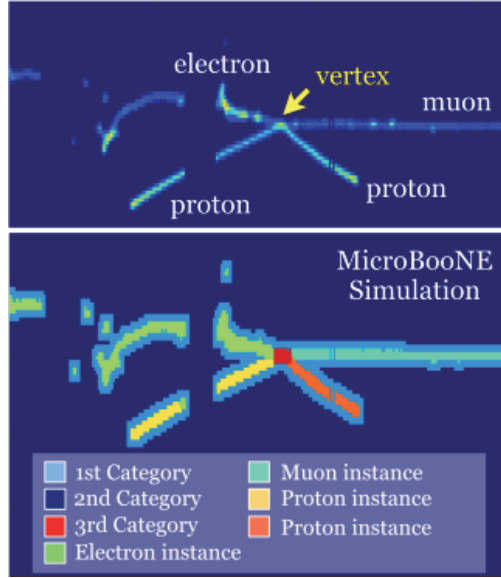


Fig. 13. Example image illustrating the pixel-wise weights used during training. (Top) Image showing the pixel values along with labels of the different particles created from a common start vertex. (Bottom) the classes of pixels to which weights were assigned to adjust their importance during training. This was done by weighting their contributions up or down relative to other pixels when calculating the average classification loss per pixel. Also, color labels indicating the particle type of certain pixels in the image. The first category of pixels that were up-weighted are pixels at the boundary of two classes or one class and background. The second class of pixels are down-weighted and consist of background pixels, defined as pixels with a threshold below some value. The third category of pixels are those at the generation vertex. These are given the largest up-weight. Figure taken from [41].

images, were prepared for both categories. This provided the events to compare the network behavior on between data and MC.

Comparisons of the track and shower score distributions between data and MC samples showed good agreement. Additionally, the data and MC agreement was checked by comparing against human-generated labels on both data and MC images. The goal was to quantify how similar was the network behavior on real and simulated images through the fraction of pixels where the network and human-labels agreed. For both the Michel electron and $CC\pi^0$ sample,

the fraction of pixels that the network disagreed with the hand-scan labels were within statistical error.

Finally, qualitative checks of network behavior were performed using ablation studies. The network score was checked on image regions where physics-based correlations are expected. These were (1) a portion of a MIP track, (2) the Bragg peak of a muon, and (3) near the trunk of a shower. All of these locations have information nearby that one would expect should correlate with a track or shower decision. For the MIP track segment, one would expect the confidence in a track-like label would increase the longer the track segment was. The Bragg peak is a very characteristic feature of track trajectories, so one would expect little change to the label regardless of information around it. And finally, for the pixels near the shower trunk, one would expect that as more pixels can be seen that branch and deviate from the initial line-like trajectory of the trunk, the confidence the trunk should be labeled as shower would get higher. All of this behavior was seen. These studies, of course, cannot guarantee that some non-physical features have influence on the track-shower determination. However, it does show that pixel patterns associated with well-known physical phenomenon are correlated with the network score. The studies also demonstrated how CNNs can assign labels to a given pixel using information from the surrounding region.

7. Concluding Thoughts

In this chapter we provided a brief survey of the ML techniques used to identify both individual particles and interaction types relevant to neutrino experiments. There are many new approaches being explored on a variety of detector types which each have different forms of information for identifying particles and interactions. Furthermore, recent developments in ML algorithms has allowed particle and interaction identification to occur at all levels of event reconstruction. In particular, algorithms like CNNs aim to exploit correlations in representations of the data close to the raw output of the detector's sensors.

Despite the wide variety of detectors and techniques applied, common issues arise. The first is that the adoption of new ML techniques sometimes imposes a representation of the data. For example, the use of CNNs require one to come up with projections into a 2D array for detectors where the spatial geometry is not naturally grid-like. Current and future work will explore how other classes of algorithms, such as graph neural networks, might be a better fit for this data.

In assembling the training data, choices are made as to the distribution of the examples over important quantities like the particle type or energy. Common approaches are to generate training data similar to the interactions expected in the detector or to generate data over distributions agnostic to the underlying physics as possible. What is the best approach is still up to debate. The latter might be more preferable for situations where models have a lot of uncertainty. But one would expect that, with all things being equal, generating the training the sample closer to the expected data distribution would be important in achieving optimal results in the context of a given analysis. Future work might include developing techniques to quantify how much the data one is running inference on is compatible with the domain of the training set. One can also modify the training of the networks to incorporate insensitivity to these prior assumptions.

The other issue is training on simulated data. Many examples shown above, use physics-based simulations to generate training data. This provides some advantages. For supervised learning tasks, simulations provide the means to associate a labeled data for just about any quantity. This has aided in the adoption of machine learning techniques in neutrino and particle physics. Of course, the drawback is that the simulations must be sufficiently similar to real data from the detectors. How to quantify what sufficiently close is is difficult. Relatedly, uncertainties in the behavior of networks needs to be quantified for different aspects of data/MC agreement. One can see various efforts above use simple manipulations of the data, such as scaling of amplitudes to make first estimates. Important future work will be needed to perform more sophisticated estimations. Here, one possible direction was seen in the NEXT example. In this case,

performance of the network was correlated with physical processes in the simulation. This again uses the access to simulations as a strength. Potential future work could exploit such connections to quantify systematic uncertainties. For example, one could estimate the variation in network behavior to values of parameters for models of physics processes.

References

- [1] Particle Data Group, Review of particle physics, *Phys. Rev. D.* **98** (2018) 030001; doi: 10.1103/PhysRevD.98.030001.
- [2] T. Katori and M. Martini, Neutrino–nucleus cross sections for oscillation experiments, *J. Phys. G* **45**(1) (2017) 013001.
- [3] R. Winyard, J. Lutkin and G. McBeth, Pulse shape discrimination in inorganic and organic scintillators. I, *Nucl. Instrum. Methods Phys. Res. Sect. A* **95**(1) (1971) 141–153.
- [4] J. Griffiths, S. Kleingesse, D. Saunders, R. Taylor and A. Vacheret, Pulse shape discrimination and exploration of scintillation signals using convolutional neural networks (2018); arXiv:1807.06853.
- [5] Y. Abreu *et al.*, Solid: A short baseline reactor neutrino experiment (2020); arXiv:2002.05914.
- [6] P. Huber, Determination of antineutrino spectra from nuclear reactors, *Phys. Rev. C* **84**(2) (2011) 024617.
- [7] S. Yousefi, L. Lucchese and M. Aspinall, Digital discrimination of neutrons and gamma-rays in liquid scintillators using wavelets, *Nucl. Instrum. Methods Phys. Res. Sect. A* **598**(2) (2009) 551–555.
- [8] L. v. d. Maaten and G. Hinton, Visualizing data using *t*-SNE, *J. Mach. Learn. Res.* **9** (2008) 2579–2605.
- [9] Y. Abe *et al.*, Reactor $\bar{\nu}_e$ disappearance in the Double Chooz experiment, *Phys. Rev. D* **86**(5) (2012) 052008.
- [10] H. de Kerret *et al.*, First Double Chooz θ_{13} measurement via total neutron capture, *Nature Phys.* **16**(5) (2020) 558–564.
- [11] A. Hoecker *et al.*, TMVA — toolkit for multivariate data analysis (2007); arXiv:physics/0703039.
- [12] R. Brun and F. Rademakers, Root — an object oriented data analysis framework, *Nucl. Instrum. Methods Phys. Res. Sect. A* **389**(1–2) (1997) 81–86.
- [13] E. Racah *et al.*, Revealing fundamental physics from the Daya Bay neutrino experiment using deep neural networks, in *2016 15th IEEE Int. Conf. Machine Learning and Applications (ICMLA)* (2016), pp. 892–897.
- [14] F. An, *et al.*, The detector system of the Daya Bay reactor neutrino experiment, *Nucl. Instrum. Methods Phys. Res. Sect. A* **811** (2016) 133–161.
- [15] A. Abhishek, W. Fedorko, P. de Perio, N. Prouse and J. Z. Ding, Variational autoencoders for generative modelling of water cherenkov detectors (2019); arXiv:1911.02369.

- [16] A. D. Missert *et al.*, T2K Collaboration, Improving the T2K oscillation analysis with fitQun: a new maximum-likelihood event reconstruction for Super-Kamiokande, *J. Phys. Conf. Ser.* **888** (2017) 012066.
- [17] S. Fukuda *et al.*, The Super-Kamiokande detector, *Nucl. Instrum. Methods Phys. Res. Sec. A* **501**(2–3) (2003) 418–462.
- [18] K. Abe *et al.*, Letter of intent: The Hyper-Kamiokande experiment — detector design and physics potential, preprint (2011); arXiv:1109.3262.
- [19] A. Li, A. Elagin, S. Fraker, C. Grant and L. Winslow, Suppression of cosmic muon spallation backgrounds in liquid scintillator detectors using convolutional neural networks, *Nucl. Instrum. Methods Phys. Res. Sect. A* **947** (2019) 162604.
- [20] M. Gell-Mann, P. Ramond and R. Slansky, in *Supergravity*, eds. P. van Nieuwenhuizen and D. Z. Freedman, (North-Holland, Amsterdam, 1979), p. 315.
- [21] T. Yanagida, in *Proceedings of the Workshop on the Unified Theory and the Baryon Number in the Universe*, KEK Report No. 79–18, eds. O. Sawada and A. Sugamoto, (1979), p. 95.
- [22] R. N. Mohapatra and G. Senjanovic, *Phys. Rev. Lett.* **44**(912) (1980).
- [23] A. Gando *et al.*, Search for majorana neutrinos near the inverted mass hierarchy region with KamLAND-Zen, *Phys. Rev. Lett.* **117**(8) (2016) 082503.
- [24] A. Gando *et al.*, Measurement of the double- β decay half-life of ^{136}Xe with the KamLAND-Zen experiment, *Phys. Rev. C* **85**(4) (2012) 045504.
- [25] M. Acero *et al.*, New constraints on oscillation parameters from ν_e appearance and ν_μ disappearance in the NOvA experiment, *Phys. Rev. D* **98**(3) (2018) 032012.
- [26] A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa and P. Vahle, A convolutional neural network neutrino event classifier, *J. Instrum.* **11**(09) (2016) P09001.
- [27] I. Esteban, M. Gonzalez-Garcia, A. Hernandez-Cabezudo, M. Maltoni and T. Schwetz, Global analysis of three-flavour neutrino oscillations: Synergies and tensions in the determination of θ_{23} , δ_{CP} , and the mass ordering, *J. High Energy Physics.* **2019**(1) (2019) 1–35.
- [28] I. Esteban, M. Gonzalez-Garcia, M. Maltoni, T. Schwetz and A. Zhou, The fate of hints: updated global analysis of three-flavor neutrino oscillations (2020); arXiv:2007.14792.
- [29] A. Diaz, C. Argüelles, G. Collin, J. Conrad and M. Shaevitz, Where are we with light sterile neutrinos? *Phys. Rep.* **884** (2020) 1–59.
- [30] F. Psihas, E. Niner, M. Groh, R. Murphy, A. Aurisano, A. Himmel, K. Lang, M. D. Messier, A. Radovic and A. Sousa, Context-enriched identification of particles with a convolutional network for neutrino events, *Phys. Rev. D* **100**(7) (2019) 073005.
- [31] J. C. Dunn, A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters (1973).
- [32] N. Yahlali, M. Ball, S. Cárcel, J. Díaz, A. Gil, J. G. Cadenas, J. Martín-Albo, F. Monrabal, L. Serra and M. Sorel, NEXT: Neutrino experiment with high pressure xenon gas TPC, *Nucl. Instrum. Methods Phys. Res. Sect. A* **617**(1–3) (2010) 520–522.

- [33] M. Redshaw, E. Wingfield, J. McDaniel and E. Myers, Mass and double-beta-decay Q value of Xe-136, *Phys. Rev. Lett.* **98** (2007) 053003; doi: 10.1103/PhysRevLett.98.053003.
- [34] J. Martín-Albo *et al.*, Sensitivity of NEXT-100 to neutrinoless double beta decay, *J. High Energy Phys.* **2016**(5) (2016) 1–30.
- [35] J. Renner *et al.*, Background rejection in NEXT using deep neural networks, *J. Instrum.* **12**(01) (2017) T01004.
- [36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2015), pp. 1–9.
- [37] R. Acciarri *et al.*, Design and construction of the microboone detector, *J. Instrum.* **12**(02) (2017) P02017.
- [38] E. Buckley *et al.*, A study of ionization electrons drifting over large distances in liquid argon, *Nucl. Instrum. Methods Phys. Res. Sect. A* **275**(2) (1989) 364–372.
- [39] P. Abratenko *et al.*, Measurement of space charge effects in the MicroBooNE LArTPC using cosmic muons (2020); arXiv:2008.09765.
- [40] R. Acciarri *et al.*, Convolutional neural networks applied to neutrino events in a liquid argon time projection chamber, *J. Instrum.* **12**(03) (2017) P03011.
- [41] M. Collaboration *et al.*, Deep neural network for pixel-level electromagnetic particle identification in the MicroBooNE liquid argon time projection chamber, *Phys. Rev. D* **99**(9) (2019) 092001.
- [42] O. Ronneberger, P. Fischer and T. Brox. U-net: Convolutional networks for biomedical image segmentation, in *Int. Conf. Medical Image Computing and Computer-Assisted Intervention* (2015), pp. 234–241.

Chapter 15

Sequence-Based Learning

Rafael Teixeira de Lima
*SLAC National Accelerator Laboratory,
2575 Sandhill Rd, Menlo Park, CA, 94025, USA
rafaeltl@slac.stanford.edu*

Sequence-based modeling broadly refers to algorithms that act on data that is represented as an ordered set of input elements. In particular, machine learning algorithms with sequences as inputs have seen successful applications to important problems, such as natural language processing (NLP) and speech signal modeling. The usage of this class of models in collider physics leverages their ability to act on data with variable sequence lengths, such as constituents inside a jet. In this chapter, we explore the application of recurrent neural networks (RNNs) and other sequence-based neural network architectures to classify jets, regress jet-related quantities, and build a physics-inspired jet representation, in connection to jet clustering algorithms. In addition, alternatives to sequential data representations are briefly discussed.

1. Introduction

Sequence-based learning deals with the concepts and algorithms used to learn from data represented as an ordered set (sequence) of objects, each with its set of characteristics (features), in which positional information of each object (context) is important. The idea of contextual information as being important for the algorithms is fundamental, since it can encode correlations between objects along the sequence. One of the main applications of this class of models is in natural language processing (NLP). In these cases, the sequence is often built from words in a sentence and the algorithm must learn

from it. How the learning occurs and what is learned will depend on the application. To perform a translation task (neural machine translation), for example, the algorithm must output another sequence. In other instances, a summary semantic information needs to be obtained, such as when the algorithm needs to classify a certain sentence as positive or negative in an online product review.

In more mathematical terms, sequence-based models aim to perform operations f on a sequence of inputs $\{\mathbf{x}^t\}$, where each entry \mathbf{x}^t is a vector of features, and t is a position in the ordered sequence with a length T , as shown in the scheme presented in Fig. 1. In particle physics terms, the sequence can represent an ordered set of tracks that constitutes a jet, for example, while the entry \mathbf{x}^t represents the kinematics of the track in the position t in that sequence. An algorithm acting on the sequence $J = \{\mathbf{x}^t\}$ can then be used to learn information about that jet, such as its flavor or charge (as will be discussed in the following sections).

In general, different sequences being utilized for the algorithm definition will have different lengths, just as jets can have different number of tracks. If all sequences have fixed length, they can be collapsed into a single feature vector and simpler algorithms, such as densely-connected NNs, can be used. However, even for fixed-length sequence problems, sequence-based models can outperform simpler models by exploiting the ordered nature of the input data.

A key feature of sequence-based models is the ability to share parameters in different parts of the same model, i.e. the operation f , which is learned, is applied to every step in the sequence. With parameter sharing, these models are able to generalize to sequences of different lengths, using the same set of parameters throughout the input elements. If different parameters were to be learned individually, the desired generalizability would not be achievable, and the model would behave similar to a densely-connected network.

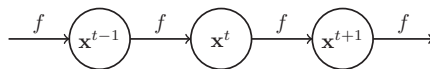


Fig. 1. Scheme of a sequence-based algorithm acting on a sequence of inputs \mathbf{x}^t .

1.1. Recurrent neural networks

A recurrent neural network (RNN) is a neural network implementation of the concepts described in the previous section. Densely-connected neural networks map an input vector of features \mathbf{x} to an output vector \mathbf{o} . In contrast, RNNs map a sequence of inputs \mathbf{x} into an output, which can be a vector or a sequence as well. This difference can be achieved in many different ways, but RNN architectures generally present cyclical connections between units in the same or different layers.

These interconnections between units are sequential, in the sense that each unit's hidden state is obtained by a combination of the previous unit's hidden state and the input from that step. This means that instead of a unit's hidden state be given by $h = f(\mathbf{x}; \theta)$, it will be given by $h^t = f(h^{t-1}, \mathbf{x}^t; \theta)$. On the particular case in which f is given by a hyperbolic tangent function, for example, the RNN can be represented as

$$h^t = \tanh \left(\mathbf{W}^\top \mathbf{x}^t + \mathbf{V}^\top h^{t-1} + b \right), \quad (1)$$

instead of a simple densely-connected unit $h = \tanh(\mathbf{W}^\top \mathbf{x} + b)$, where \mathbf{W} , \mathbf{V} , and b represent learnable weights and biases.

Notice that in Eq. (1) the weights on the operations (θ , or \mathbf{W} , \mathbf{V} , and b explicitly) do not depend on the time step, or the sequence element position, t , explicitly showcasing the parameter sharing feature of the RNN. Similarly to MLPs functioning as universal approximators, large enough RNNs have been shown to universally approximate any measurable sequence-to-sequence maps [1].

In general, RNN architectures add other features on top of the recurrent layer format described above. For example, in tasks where the algorithm needs to output another sequence, each individual hidden state in the recurrent layer might be read out into a densely connected network. In contrast, the cases where only a single output is read at the end of the sequential layer (also known as time-unfolded RNNs) are used to extract a summary information of the input sequence. Even though the presence of cycles in these architectures could potentially complicate the process of updating the

network parameters during the optimization step, backpropagation can still be applied to the unrolled computation graph,^a thus no specialized algorithms are necessary.

A simple but powerful extension of standard RNN architectures are bidirectional recurrent neural networks (BRNNs) [2]. For certain sequence-based modeling applications, knowledge about backward-in-time context might be as important as forward-in-time, only the latter of which is exploited in standard RNNs. BRNNs manage to extend that context information by adding a second recurrent layer to the network architecture which processes the sequence in reversed order. Both the forward and backward RNNs are then connected to the same output layer, providing a combined representation.

An idea related to RNN architectures which is also used in time-series and signal processing analyses are 1D convolutional layers. In these architectures, the convolution operation kernel acts on neighbouring time steps, and outputs a new sequence based on its inputs. These operations act on fixed length sequences, where empty entries can be masked — similarly to 2D CNNs acting on sparse images with a fixed pixel grid. Parameter sharing is also an important feature here, exploited by the use of a single convolution operation across different time steps. One drawback of this method is its limited sensitivity to long-term dependencies, only exploring correlations across close neighbours, as defined by the kernel length.

1.2. Long-term dependencies, *LSTMs* and *GRUs*

When dealing with large sequences, one expected behavior of RNNs is to learn how correlated certain entries are, regardless of how far apart they appear in the input sequence. In practical terms, this implies that information from early entries in the sequence must be encoded in how the network learns about latter entries (long-term dependencies). Unfortunately, in the learning steps, it is common for gradients propagated through many steps to either quickly approach zero or increase — both undesirable features in terms of optimization.

^aThe application of backpropagation on the time-series unrolled RNN is known as *backpropagation through time*.

This issue is referred in the literature as the *vanishing or exploding gradient problem* (e.g. in [3]). Vanishing gradients, for example, can lead to long-term dependencies being given less importance through the sequence compared to short-term ones.

This issue can be understood by imagining a simplified recurrent structure as a linear transformation between hidden states $h_t = \mathbf{V}^\top h_{t-1}$. This operation will be therefore performed T times when moving from the time-step 0 to the last sequence entry. This shows that the learnable parameters in \mathbf{V} will be raised to the power of T , which means that weights less than 1 will tend to approximate 0 at later steps, while weights larger than 1 will quickly grow. Most modern RNN architectures solve this problem with the usage of long-short-term memory (LSTM) [4] units or gated recurrent units (GRUs) [5].

LSTM units mitigate the vanishing gradient problem by introducing a new path in the recurrent loop where the information coming from each sequence entry can flow for long durations, possibly without interference from subsequent hidden states. This path is dynamically gated through learnable parameters, which means that the importance of long-term dependencies is optimized with the rest of the network parameters. In particular, if the activation function pertaining to a gate remains close to 0, the information from the previous time-step will not be propagated throughout the sequence. However, the LSTM will still use that time-step information to update its current hidden state. This optimizable gated structure ensures that both long-term and short-term contributions to the gradient are taken into account.

A schematic view of three sequential LSTM units is shown in Fig. 2. The LSTM receives at the time step t both the hidden state of the previous time step (h_{t-1}), which is concatenated with the feature vector \mathbf{x}_t , and an extra input (the cell state, C_{t-1}) which is regulated by a forget gate (f_t). The forget gate can be a neural network itself, with a sigmoid output which is shared across all units, ensuring that the LSTM actively learns how much long-term correlations should be propagated in the sequence. Next, the cell state is updated with information learned from the previous hidden state and \mathbf{x}_t with a

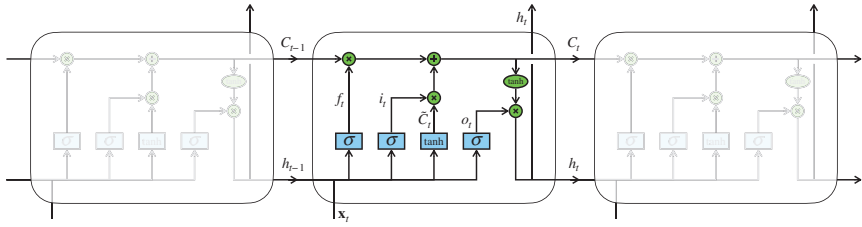


Fig. 2. Schematic structure of an LSTM recurrent layer. Two outputs for the unit's hidden state h_t are shown, representing the case in which the LSTM layer outputs another sequence. Image adapted from [6].

dedicated neural network, generating an intermediate cell state \tilde{C}_t . The impact of \tilde{C}_t in the final cell state is regulated by another neural network, i_t . After the update, C_t is propagated to the next time-step. The final hidden state at this time step is derived with information learned from the previous hidden state and \mathbf{x}_t through a network (o_t), and also \tilde{C}_t .

GRUs work with a similar gated structure — however, the same forget gate that decides on the propagation of the previous cell state also determines, with an inverse importance, how the current cell state should be updated with h_{t-1} and \mathbf{x}_t . With the LSTM nomenclature used above, the forget gate and the cell state update gate would be given by $f_t = u_t$ and $i_t = (1 - u_t)$, respectively, where u_t is called the update gate. GRUs comparatively uses less trainable parameters than LSTMs, which can be beneficial for smaller datasets; its less flexible architecture can potentially be detrimental in more complex applications — however, the studies to be shown below that compared GRUs and LSTMs generally see similar performances.

2. Applications of RNNs to Jet Physics

Representing reconstructed jets in collider experiments as images for classification and other machine learning-based tasks, has been a successful avenue of research for years now (see Chapter 13). There are, however, a few features related to this choice of representation that can make the process of training a computer vision-based algorithm difficult when compared, for example, to the training of a simple densely-connected neural network based on engineered features.

Jet images can be very sparse, i.e. containing few populated pixels,^b making the identification of features on individual jets a complicated task, even if identifying these features on their averaged images might be very easy. Pre-processing steps can be applied to reduce sparseness, for example, increasing the coarseness the image further by combining adjacent pixels. This procedure, however, effectively reduces the spatial information contained in the input image, penalizing the algorithm's performance.

Another issue arising in the pre-processing step is finding a unique geometrical representation for these images that reflects the expected symmetries of the problem. In general, geometrical transformations (e.g. rotations, translations and reflections) are used, aligning pre-defined axes based on the jets' spatial energy distributions — but these definitions can be very task specific and not well generalizable. This is specially true when the pattern of energy deposition inside these jets displays a different number of core clusters (prongness) — for example, when comparing quark-initiated jets, jets from a collimated hadronic W-boson decay, and jets from a collimated top quark decay.

The algorithms to be described below think about the jet instead as collection of correlated objects, each one with its set of characteristics (such as position and energy), and apply some of the sequence-based ML ideas discussed above for different types of tasks. This idea is reminiscent of the actual way jets are built in collider experiments, using sequential clustering algorithms (for example, the widely used in LHC experiments anti- k_t algorithm [8]), acting on low-level detector quantities, such as calorimeter deposits or tracks.

2.1. Identifying heavy flavor jets

The identification of jets originated from the products of the hadronization process of *heavy* quarks (bottom and charm quarks), known as heavy flavor jets, is of fundamental importance for experiments at the LHC for two main reasons. Firstly, the Higgs boson — discovered in 2012 by ATLAS and CMS, and currently the focus

^bInitial computer vision-based jet images studies have reported 5–10% of activated pixels on average for 25×25 pixels images from signal- and background-type jets [7].

of intense experimental investigation — mainly decays to a pair of bottom quarks, with a predicted decay branching fraction of about 56% [9]. Secondly, the top quark, which together with the Higgs boson can help us learn about the structure of the electroweak vacuum [10], decays almost entirely to a bottom quark plus a W-boson [11].

Finding these Higgs and top decays is not an easy task due to the enormous *multijet* background events at the LHC (events with at least two reconstructed jets). These multijet events are produced with cross-sections over three times larger than events with top quarks, and four times larger than events with a Higgs boson [12]. Fortunately, most of these events contain *light* jets, which are originated by up, down, strange quarks and gluons. Therefore, learning how to separate heavy flavor signal jets from the light jets background is necessary.

Hadrons containing bottom and charm quarks are heavy — for example, the B^0 meson invariant mass is about $5.3 \text{ GeV}/c^2$ [11]. They decay through the Weak force, thus having a long mean lifetime ($\tau_{B^0} \simeq 1.5 \times 10^{-12} \text{ s}$). Therefore, a B^0 with a momentum of $50 \text{ GeV}/c$ will travel on average over 4.5 mm before decaying. This displaced decay can be reconstructed as a *secondary vertex*, i.e. a vertex that is separated from the collision's primary vertex (where the initial proton–proton interaction occurred, in the case of the LHC). The LHC experiment's inner trackers (responsible for reconstructing the trajectory of charged particles) have been built with the intent of separating these secondary vertices with good precision, in order to identify heavy flavor jets. The decay chain of a B^0 meson, as simulated by the ATLAS experiment, is shown in Fig. 3 [13], which also includes the *tertiary vertex* from the displaced decay of the D^0 meson, containing a charm quark.

In general, requiring that a secondary vertex needs to be reconstructed in order to identify a b -jet can be detrimental for a high-efficiency algorithm. However, tracks originated from this displaced location will have very particular characteristics when compared to tracks from the primary vertex, and will be correlated by their shared origin. Therefore, algorithms that focus on finding correlations between tracks perform comparatively well with respect to direct

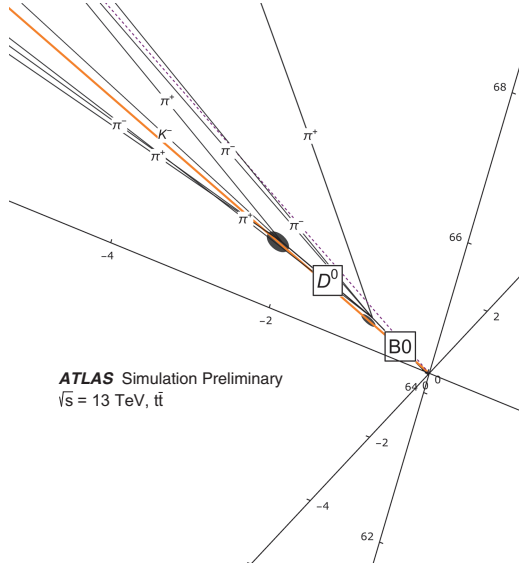


Fig. 3. Simulated decay chain of a B^0 meson in the ATLAS experiment. This event display was obtained from a simulated dataset of top quark pair production, at center-of-mass energy of 13 TeV. It shows the displacement of the vertices produced from B^0 (secondary vertex) and subsequent D^0 (tertiary vertex) decays, with respect to the center of the coordinate system (primary vertex).

secondary vertex finding, and can provide complimentary information for a combined heavy vs. light jet discrimination.

In the ATLAS experiment, two different types of algorithms have been developed to identify heavy flavor jets based on the likelihood of tracks being originated from secondary vertices. Both use the tracks' *impact parameter* information, which encodes the distance of closest approach of the charged particle's trajectory with respect to the primary vertex. Particles originated from the primary vertex will have small impact parameters, while particles from secondary vertices will tend to have larger impact parameters. An important related quantity is the impact parameter significance, in which the distance is divided by the uncertainty in its measurement. Utilizing the significance minimizes the impact of low-quality tracks with large mis-measured impact parameters.

IP3D [14], one of the first ATLAS algorithm based on tracks impact parameter information, treats the tracks as independent

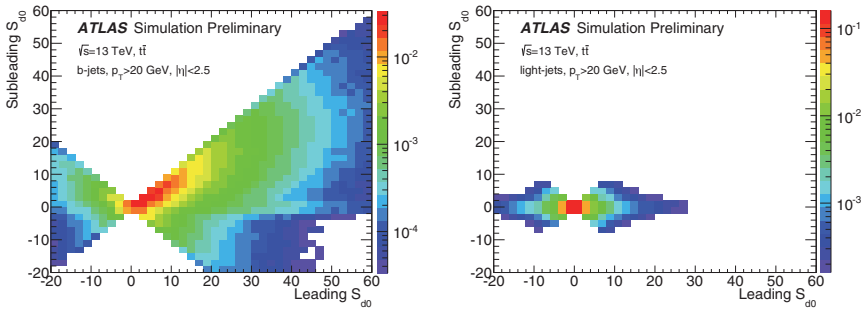


Fig. 4. 2D histogram of S_{d0} for leading (horizontal axis) and subleading (vertical axis) tracks inside a b -jet (left) and a light flavor jet (right). The correlation observed is an indication that the naïve Bayes approach of the IP3D algorithm is not enough to exploit the full information contained in the tracks' impact parameters with respect to b -jet identification.

entities, ignoring possible correlations. It uses 3D histograms of transverse and longitudinal impact parameter significances (S_{d0} and S_{z0} , respectively), and a track quality grade, built from simulation and separately for b -jets, c -jets and light flavor jets. Per-flavor conditional likelihoods are calculated from these histograms for each track in a jet. With a naïve Bayes approach, a final jet-level likelihood is built by multiplying the individual tracks likelihoods. However, important information is lost with the assumption that the tracks in the jet are uncorrelated. This can be seen in Fig. 4, where a strong correlation between the S_{d0} distribution of the leading and subleading track in the jet (ordered by S_{d0}) can be seen near the diagonal for b -jets but not for light jets.

More recently, algorithms exploiting RNN architectures based on LSTMs have been proposed to perform the task described above, treating the list of tracks within the jet as the input sequence to the algorithm. Due to the variable number of tracks within a jet, RNNs are better suited than dense architectures in this approach. Even though no natural track ordering is clear to the problem, tracks with larger impact parameters are more likely to come from heavy flavor jets, therefore, impact parameter based ordering is a good ansatz.^c

^cEmpirically, the studies mentioned below have shown that certain track orderings work better than others.

While the most basic version of this algorithm acts on tracks only, proposals have been made to combine track and secondary vertices information in a single LSTM-based architecture [15].

The ATLAS implementation of the LSTM-based architecture for heavy jets identification with tracks' impact parameters is called RNNIP [16]. It treats the tracks within a jet as a sequence, and uses the impact parameter significance information and track kinematics as features. It also uses categorical information based on the track reconstruction quality in an embedded layer. These categories separate high quality, well measured tracks, in which a better impact parameter resolution is expected, based on detector-level information such as the number of hits in the innermost tracker layer.

The tracks are ordered by \mathcal{S}_{d_0} , although other orderings (such as by track p_T) have shown similar performance. The algorithm is trained in a simulated sample of top pairs, which provide a dataset enriched of both heavy and light quarks, and outputs a probability of a given jet to be a bottom jet (b -jet), a charm jet (c -jet), or a light flavor jet. The three probabilities are then combined into a likelihood that is used for discrimination.

A comparison between the performance of the naïve Bayes algorithm described above (IP3D) and RNNIP is shown in Fig. 5. The efficiency of identifying b -jets is plotted on the horizontal axis, while one over the probability of identifying light flavor jets (misidentification probability) as b -jets is plotted on the vertical axis. The RNNIP algorithm displays a better light flavor jet discrimination for every value of b -jet efficiency, and is comparable to a boosted decision tree that combines IP3D with the secondary vertex-based ATLAS algorithms (MV2c10 [14]), even though it does not explicitly reconstruct secondary vertices. Performances were measured for jets clustered with the anti- k_T algorithm [8] with $R = 0.4$, with a transverse momentum above 20 GeV, in a simulated dataset of top quark pairs, at center-of-mass energy of 13 TeV.

Figure 6 shows the Pearson's correlation coefficient ρ between the RNNIP likelihood, and \mathcal{S}_{d_0} and \mathcal{S}_{z_0} for each track in the sequence. It is interesting to note that stronger correlations in b -jets are seen for impact parameter significances of the first ~ 8 tracks, which may be related to the expected charged particle multiplicity of b -hadron

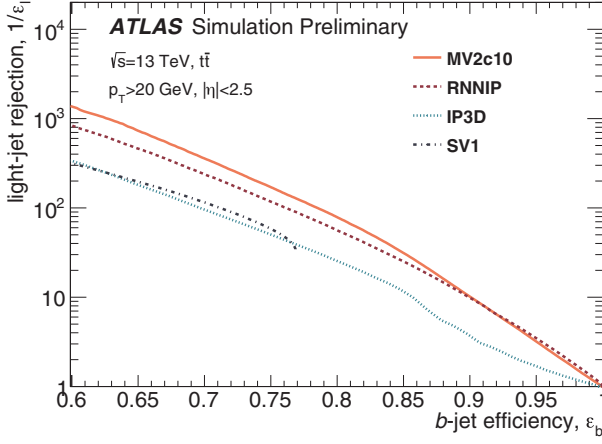


Fig. 5. Performance of heavy flavor identification algorithms in the ATLAS experiment [16]. The horizontal axis shows the efficiency of correctly identifying b -jets, while the vertical axis shows one over the efficiency of incorrectly identifying light flavor jets as b -jets. The red dashed curve shows the performance of the RNN-based algorithm (RNNIP), while the dashed blue curve shows the performance of a naïve Bayes algorithm acting on similar inputs (IP3D).

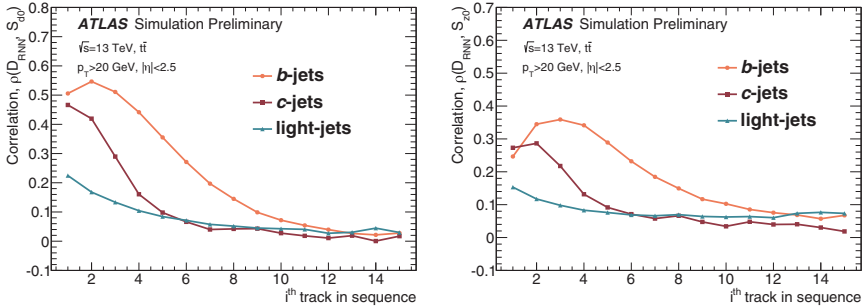


Fig. 6. Pearson's correlation factor between RNNIP likelihood, and transverse and longitudinal impact parameter significances. Correlations are shown separately for b -jets (orange), c -jets (red), light flavor jets (blue).

decays. This shows that the network architecture is able to learn contextual information from the given sequence ordering.

Heavy flavor jets identification in the CMS experiment shares many similarities with the strategies employed by ATLAS. In particular, their final discriminant is also a combination of information

pertaining to secondary vertexing and the set of tracks inside the jet. Two sets of algorithms have been developed with this intent: one set combining engineered features extracted from the jet, and one that directly uses reconstructed objects information into a neural network architecture which includes LSTM layers. The CMS DeepCSV algorithm [17] exemplifies the first strategy, similarly to the ATLAS MV2c10 boosted decision tree. It is based on a densely-connected neural network with four hidden layers, with inputs that are defined by other algorithms which act directly on tracks' impact parameter information and reconstructed secondary vertices.

Significant improvement has been observed by CMS by moving to the DeepFlavor algorithm [18], which acts directly on these low-level observables. The DeepFlavor network receives three sequences as inputs: a sequence of charged particles (reconstructed from tracks and calorimeter clusters), a sequence of neutral particles (calorimeter clusters with no associated tracks), and a sequence of reconstructed secondary vertices. Each sequence is processed by a one-dimensional (1D) convolutional layer, which learns a shared representation that is specific for each type of sequence. The convolutional layer outputs are then fed to three different LSTM layers, which summarized the sequences information into three fixed length feature vector. These features are combined with additional jet-level information in a densely connected network, which outputs the jet flavor probabilities.

Figure 7 summarizes the CMS b -jet identification performance. The red and blue lines represent the performance of the DeepCSV and DeepFlavor algorithms respectively, with the CMS detector conditions present during the 2017 LHC data taking period (Phase 1), while the green line shows the DeepCSV performance with the CMS detector conditions in 2016 (Phase 0). Between 2016 and 2017, CMS inner tracking detector was upgraded to deal with the harsher radiation conditions at the LHC later Run 2 years. This upgrade also provided the CMS experiment a better impact parameter resolution, directly improving their heavy flavor identification performance. The b -jet efficiency is shown as a function of the light flavor jets misidentification probability (full lines), and as a function of the c -jets misidentification probability (dashed lines). While a large improvement is seen in the performance of the DeepCSV algorithm with the CMS

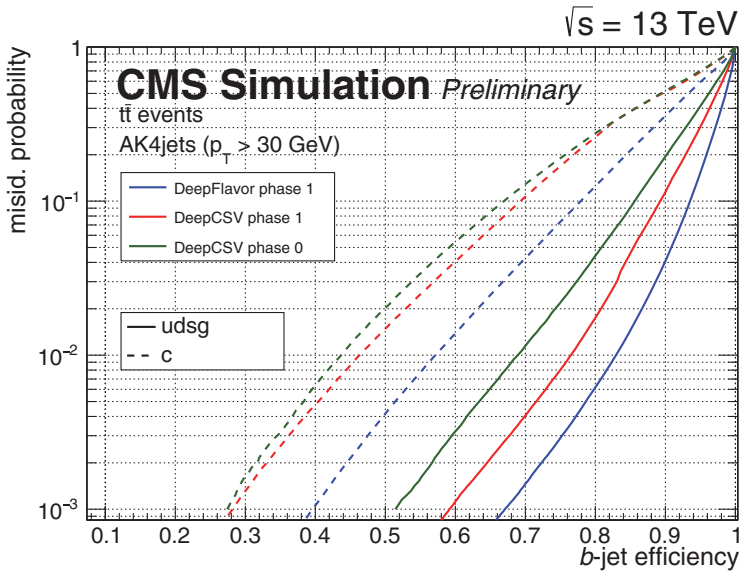


Fig. 7. Performance of heavy flavor identification algorithms in the CMS experiment [18]. The horizontal axis shows the efficiency of correctly identifying b -jets, while the vertical axis shows the efficiency of identifying light flavor jets (full lines) or c -jets (dashed lines) as b -jets. The red and blue curves represent the DeepCSV and DeepFlavor algorithms performances with the 2017 CMS detector conditions, while the green curve represents the DeepCSV algorithm with the 2016 CMS detector conditions.

Phase 1 inner tracker upgrade, an improvement just as large is seen with the usage of DeepFlavor in terms of light flavor jet discrimination, with an even large gain in terms of c -jet rejection. Performances were measured for jets clustered with the anti- k_T algorithm, with a transverse momentum above 30 GeV, in a simulated dataset of top quark pairs, at center-of-mass energy of 13 TeV.

2.2. Identifying strange jets

Jets from strange quarks are grouped within the light flavor jets category for the algorithms described above. However, for certain physics applications, such as the direct measurement of the $|V_{ts}|$ element of the CKM matrix through the search of the rare decay $t \rightarrow W^+ s$

(or $\bar{t} \rightarrow W^- \bar{s}$) [19], discriminating strange jets from first-generation jets is a necessity.

A LSTM-based algorithm has recently been proposed to tackle this problem [20]. The study is performed with a simplified detector description model (Delphes [21]) based on a CMS-like detector. The algorithm is trained to discriminate between strange jets and jets from the hadronization of up and down quarks, produced by proton–proton QCD interactions. Similarly to the ATLAS RNNIP, the proposed algorithm acts on sequences of tracks, with features based on their impact parameters and kinematics with respect to the jet.

Secondary vertices are expected to be present in strange jets through the decay of strange kaons into $\pi\pi$ and lambda baryons into $p\pi$. Therefore, all possible secondary vertices in the jet are reconstructed by pairing tracks with small distances of closest approach to each other. These vertices are then used to define a track ordering based on the parameter R assigned to each track. This parameter is defined either by the transverse distance between the primary vertex and the secondary vertex to which that track belongs, or, in the cases where the track is not associated to a secondary vertex, the innermost tracker hit belonging to that track. If multiple tracks receive the same R (e.g. two tracks from the same secondary vertex), their ordering is performed by p_T . This ordering ensures that adjacent tracks belong to the same secondary vertex, which the network will use to learn about displaced decays.

Figure 8 compares the performance of the LSTM-based strange jet discriminator to the performance of simpler methods. In particular, the performance of using the transverse momentum fraction x_K and x_Λ of identified kaons and lambda baryons is also investigated. To calculate these quantities, a selection on the invariant mass of the reconstructed secondary vertices, consistent with the K and Λ masses, is applied. The highest p_T K and Λ candidates of the remaining vertices are chosen, and used to calculate x_K and x_Λ to their parent jet. Two different setups of the LSTM are compared: one including all selected jets, and one only including jets that contain at least one track with large transverse impact parameter $|d_0| > 1$ mm.

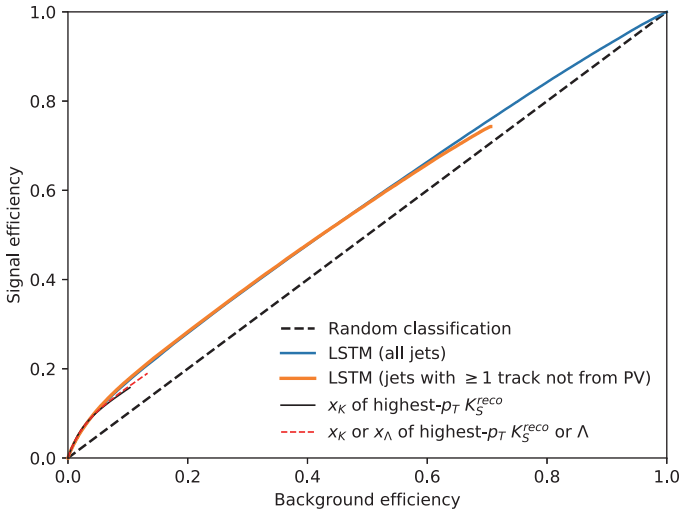


Fig. 8. Performance of strange jet identification algorithms based on LSTM architectures and strange hadron reconstruction [20]. The vertical axis shows the efficiency of correctly identifying strange jets, while the horizontal axis shows the efficiency of incorrectly identifying up and down quark jets as strange jets. The blue and orange lines show the performance of the LSTM-based algorithm using all jets and jets with at least one track with transverse impact parameter $|d_0| > 1$ mm, respectively. The red dashed and full black lines show the performance of selecting on the transverse momentum fractions x_K and x_Λ , and only on x_K , respectively.

The overall performance of this algorithm is unfortunately limited by the similarity between strange and first-generation quark jets — achieving a background efficiency of 21% (63%) for a signal efficiency of 30% (70%). The secondary vertices from kaon decays are not as displaced as vertices from b -jets or c -jets and can easily be misidentified from vertices produced by material interaction or decays of particles originated in the hadronization process. One important improvement with respect to using x_K and x_Λ , however, is the ability to achieve higher signal efficiencies, as shown in Fig. 8.

2.3. Identifying tau lepton jets

Similarly to heavy flavor jet identification, identification of tau leptons is particularly interesting due to its ties to Higgs physics.

The coupling between the Higgs and the tau lepton is the largest Higgs couplings to leptons in the Standard Model. It is therefore an opportunity to directly measure the structure of the Higgs Yukawa couplings to that sector of the Standard Model.

Taus decay either leptonically ($\tau \rightarrow \nu_\tau + \ell \nu_\ell$, in which ℓ is an electron or a muon), or hadronically ($\tau \rightarrow \nu_\tau + \text{hadrons}$). Leptonic tau decays are roughly indistinguishable from isolated leptons in hadron collider experiments. Therefore, tau identification focuses on hadronic taus, which represent a branching fraction of approximately 65% [11]. Hadronic tau decays usually include one or three charged pions and one or more neutral pions. Therefore, these decays are seen in the detector as narrow jets with one or more tracks.

Since neutral pions do not leave signals on the detectors' inner tracks, their trajectories cannot be reconstructed as tracks. This means that if only track information were used, a large portion of information for tau identification would be missing. Therefore, an optimal strategy should aim to combine the tracking and calorimetry information.

The ATLAS experiment state-of-the-art tau identification algorithm is based on a double LSTM architecture that combines track sequences and calorimeter deposits (clusters) sequences [22]. The algorithm has three sets of inputs: a track sequence, a cluster sequence, and a set of high-level variables connected to a dense layer. The track and cluster features considered refer to their kinematics and detector-level properties, while the high-level ones are related to the jet itself, or engineered features based on the collection of jet constituents.

Tracks are individually fed through dense layers with shared weights, so that an embedded representation can be learned. The same procedure is applied to calorimeter clusters separately. The two processed sequences, one of track embeddings and one of cluster embeddings, are ordered by decreasing p_T of the original objects and used as inputs to two separate LSTM blocks. These blocks includes two layers of LSTM units — the first one maps the sequence in the learned representation into another sequence of the same length. The second LSTM layer only outputs the information at the last

time-step, thus providing a summary of the input sequence. The LSTM blocks outputs are fed to a densely-connected block, which also receives information from a densely-connected block encoding high-level observables of the tau jet.

The training and evaluation of the architecture defined above is performed separately for the cases in which the tau decay includes one or three tracks. The tau decays (signal) are provided by simulation of $\gamma^* \rightarrow \tau\tau$ events, while background jets are selected from a simulation of dijet events.

The performance obtained in Fig. 9 compares the LSTM architecture (RNN) optimized for tau decays with one (1-prong) and three (3-prong) tracks. It also compares to the previous algorithm used in the ATLAS experiment, based on a boosted decision tree (dashed lines). The LSTM-based architecture outperforms the previous baseline for all hadronic tau efficiencies. These improvements have been

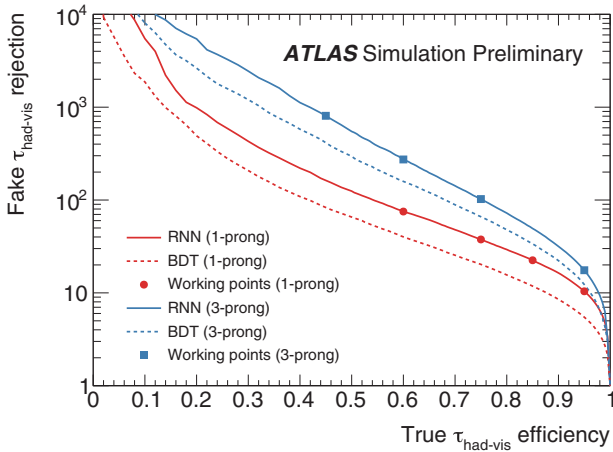


Fig. 9. Performance of hadronic tau jets algorithms in the ATLAS experiment [22]. The horizontal axis shows the efficiency of correctly identifying hadronic tau jets, while the vertical axis shows the inverse of the efficiency of incorrectly identifying quark-initiated jets as hadronic tau jets. The red curves show the performance exclusively on tau jets with a single track (1-prong), while the blue curves represent tau jets with three tracks (3-prong). The RNN-based model's performance is shown in the full lines, with the dashed lines representing an algorithm with similar inputs but based on a boosted decision tree.

shown to be significant enough that the new architecture has actually been used for identifying tau candidates at the ATLAS high-level trigger (HLT) in 2018.

2.4. Identifying top jets

When produced at large momenta, top quarks' decay products will start to merge, making it more difficult to resolve them spatially in the detector. In this regime, the entire top quark decay can be clustered into a single jet. Usually these jets have larger radius parameters: CMS [23] utilizes $R = 0.8$ anti- k_t jets and $R = 1.5$ Cambridge-Aachen jets [24], while ATLAS [25] focuses on $R = 1.0$ anti- k_t jets. Identifying these boosted top objects, from a background of jets from the hadronization of lighter quarks and gluons, is particularly interesting when searching for Beyond the Standard Model physics which predict TeV-range resonances decaying to top pairs.

Several interesting features which are present in a boosted top jet can be used to discriminate against a high momentum jet produced by QCD interactions. In particular, hadronic top decays will tend to be three-pronged, with each prong corresponding to a final state particle in the $t \rightarrow bW \rightarrow bq q'$ decay chain. Two important details on this chain is that one of these prongs will be consistent with a heavy flavor jet, and the other two will be consistent with a W-boson decay.

LSTM-based architectures have been proposed for identifying these boosted top jets [26]. The study is performed with a Delphes-based detector simulation [21] with a particle flow type of particle reconstruction, overlaying minimum bias events to emulate the LHC 2016 collisions conditions, averaging of 23 proton-proton interaction per event. Jets are clustered following the ATLAS strategy, with the anti- k_T algorithm and $R = 1.0$. The signal top jets are obtained from simulating a beyond the Standard Model process in which a Z' -boson with masses ranging from 1400–6360 GeV decays to hadronically decaying $t\bar{t}$ pairs. Background jets come from the simulation pure QCD hard scattering processes (QCD jets).

Similar to tau identification, the sequence for the recurrent model is built of calorimeter clusters. The input ordering is defined based on

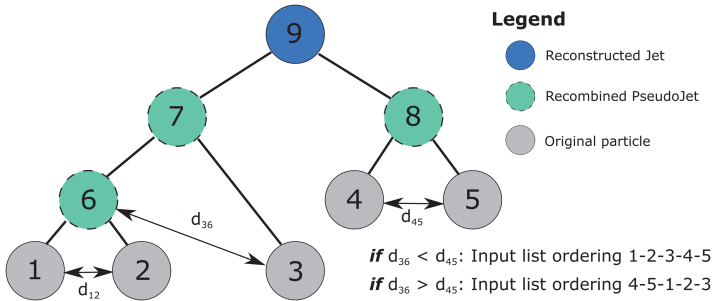


Fig. 10. Scheme of jet clustering history used to define input sequence ordering in LSTM-based top identification algorithm [26]. The anti- k_T distance d_{ij} is used to choose which path to follow in the tree.

going through the clustering history of the jet, starting from the final reconstructed jet, and adding constituents to the sequence as they appear in each step. The decision on which path to follow in each time two parent nodes merge depends on the anti- k_T distance metric d_{ij} . If the two parent nodes are present in the list of jet constituents, they are added to the sequence ordered by p_T . A scheme presenting how the clustering tree is used to build the sequence ordering is shown in Fig. 10. Another strategy is based on reconstructing $R = 0.2$ anti- k_T subjets, order them by descending p_T , then adding constituents to the sequence depending to which subjet they belong — constituents on the same subjet are also ordered by descending p_T . These schemes are compared with purely ordering on the jet constituents p_T .

As seen in Fig. 11 (left), the LSTM-based architecture with substructure ordering outperforms the previous baseline studied by the same group [27], based on a fully connected dense neural network. The results are presented depending on the ordering scheme as well whether the jets are trimmed or not. Trimming [28] is a technique that ensures robustness of the jet kinematics, especially the jet mass, with respect to pile-up contamination by removing $R = 0.2$ subjet constituents with a certain energy fraction; in this case, subjets are removed if their energy fraction is lower than 5%. Figure 11 (right) shows that the impact of the ordering strategy is limited. In fact, it appears to be smaller than the impact of applying trimming, which

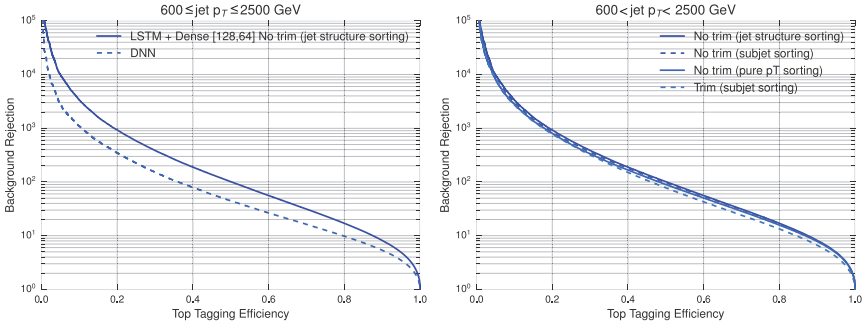


Fig. 11. Performance of top identification with LSTM-based techniques [26]. The horizontal axis shows the efficiency of correctly identifying top jets, while the vertical axis shows one over the efficiency of incorrectly identifying QCD jets as top jets. The left plot compares the LSTM algorithm with a strategy using a dense neural network. The right plot compares different training strategies for the LSTM, including different sequence orderings (substructure, subjet or purely constituent p_T based) and with or without trimming.

removes subjet information that could be important for top identification. For more information on top jet identification with jet images, see Chapter 13.

2.5. Learning a jet representation

In general, a jet can be understood as a collection of final state particles that are results of the hadronization and fragmentation processes of a quark or a gluon. However, the actual procedure of reverse-engineering these processes, which are quantum mechanical by nature, can be complicated, especially when dealing with the busy environment of a hadron collider event. Jet clustering algorithms aim to reduce this complexity and make the connection between observable final state particles and the phenomenological predictions based on partons.

While many types of jet clustering algorithms have been proposed in the past, most recent applications utilize strategies based on sequential clustering, such as the ones already discussed here (k_T , anti- k_T and Cambridge-Aachen being three well-known examples). These algorithms take advantage from the fact that the processes for

the particle shower development can be approximated well, due to factorization theorems, by a sequence of $2 \rightarrow 1$ splittings in creating a hierarchical representation of the jet. When reverse-engineered, this sequence of splittings becomes a sequence of mergings of jet constituents, forming a binary tree. This binary tree represents the jet clustering history, and encodes important information about the nature of that jet.

As presented in the previous section, the usage of RNN architectures for learning jet labels has been a successful avenue in collider experiments, greatly improving on previous, often already Machine Learning-based, strategies. Treating the jet constituents simply as an input sequence, however, does not fully exploit the full information contained in the jet clustering history as represented by the clustering tree.

It has been suggested that through understanding the interplay between the clustering history and neural network architectures could lead to a more natural jet representation [29]. The study presents a framework in which a neural network can be built based on the clustering tree obtained from the jet clustering algorithm history. It tries to exploit the entire physical knowledge contained within the jet, particularly with respect to hierarchical correlations between individual constituents. This structure is then used to learn an overall probabilistic model of the jet conditioned on its constituents, through an unsupervised training task.

Building a probabilistic model for a class of jets presents the possibility of having a tractable and differentiable distribution function which can inform different aspects of jet physics. In particular, models trained for different classes of jets, such as b -jets and light flavor jets, can be used to compute likelihood ratios for discrimination. Models can also be sampled in order to generate new jets, a process which can take a significant amount of time when large datasets are required.

The framework, named JUNIPR (“Jets from UNsupervised Interpretable PRobabilistic models”), is based on the factorization of the jet probabilistic model into a product of probabilities given by each step of the clustering tree. For a set of jet constituents denoted

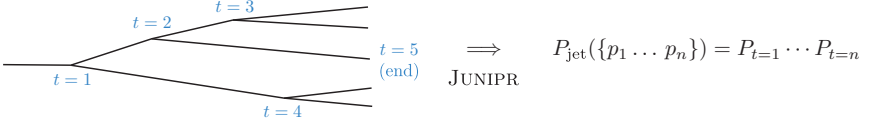


Fig. 12. JUNIPR calculation of jet-level probability model based on its factorization to nodes in binary tree of the jet clustering history [29].

by their 4-momenta p_1, \dots, p_n , it computes the probability density $P_{\text{jet}}(\{p_1, \dots, p_n\})$ of this set to have arisen from the specified model. This probability factorizes based on the $2 \rightarrow 1$ clustering tree, so that $P_{\text{jet}}(\{p_1, \dots, p_n\}) = \prod_t^{t=n} P_t$, where P_t represents the probability model of the branching step t . This factorization is schematically shown in Fig. 12.

The study further models P_t as the product of three independent probabilities: P_{end} , probability over binary values predicting if the tree stops after this split; P_{mother} , probability over integers of how likely is that the mother in step t indeed participates on the splitting $1 \rightarrow 2$; and P_{branch} , probability over kinematic configurations of the three states involved in the branching step t . Each one of these three probabilities are modeled with densely-connected neural networks. They use as inputs the hidden state h_t at branching step t , calculated based on the recurrent relation below:

$$h_t = \tanh(V \cdot (k_{d_1}^t, k_{d_2}^t) + W \cdot h_{t-1} + b), \quad (2)$$

where $k_{d_1}^t, k_{d_2}^t$ represent the 4-momenta of the two daughters involved in the splitting step t , and V , W , and b represent learnable weights and biases. This equation has the same form as the one presented in Sec. 1.1 detailing the functioning of RNNs. The study also tried to replace this simple RNN strategy in the equation above with more complex solutions, such as LSTMs and GRUs. No significant improvement in performance was observed; it was argued that this was due to the simplicity of the task, acting on sequences with only two elements.

The algorithm is trained based on the full jet probabilistic model, maximizing the log-likelihood over the examples in the training dataset, with stochastic gradient descent. Datasets are generated

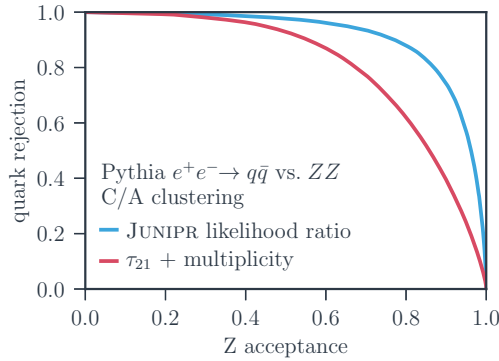


Fig. 13. Performance of JUNIPR used for binary classification, based on the calculation of likelihood ratios [29]. The horizontal axis shows the efficiency of identifying Z -jets, while the vertical axis shows one minus the efficiency of incorrectly identifying quark jets as Z -jets. The image also shows the discrimination performance by simply using a substructure variable τ_{21} and constituents multiplicity information.

by e^+e^- collisions, without detector simulation, at center-of-mass energy of 1 TeV. The two jets in the final state are separated into hemispheres by the exclusive k_T algorithm [30], and their clustering trees are obtained with the Cambridge-Aachen algorithm.

Results of using the learned jet probability applied to jet classification, through likelihood ratio computation, are shown in Fig. 13. The blue line represents the discrimination power between jets from quarks and jets from a boosted hadronic Z decay by calculating the likelihood ratio between the jets probabilities based on each hypothesis $P_Z(\text{jet})/P_q(\text{jet})$. $P_Z(\text{jet})$ and $P_q(\text{jet})$ represent the JUNIPR probabilities trained individually on Z jets and quark jets, respectively, and evaluated on a given jet. The JUNIPR likelihood ratio greatly outperforms the strategy based on engineered features representing the jet substructure.

The framework has also been used for direct binary classification tasks [31]. In this case, two JUNIPR networks are built based on two different types of jets (quark jets and gluon jets, for example). The two networks are trained with a cross entropy objective function, where the individual jet probabilities from each jet type

are defined by the JUNIPR networks. The quark vs. gluon discrimination achieved by this method was seen to outperform standard approaches, such as CNNs on jet images. It also significantly outperforms the strategy above of individually training JUNIPR models and calculating their likelihood ratio.

3. Alternatives to RNNs

The applications of RNN architectures in collider experiments and phenomenological studies has been an active area of recent developments, successfully avoiding issues previously seen with CNN architectures, while very often improving on their performances. Some features of these architectures are, however, less desirable for certain problems. Certainly the physics problems utilizing RNNs do not have a well-defined, natural ordering of the sequence elements. Choosing a specific order becomes a non-trivial step of the data formatting, one that could lead to undesirable performance losses. On the other hand, other physics problems might benefit from topological structures that encode more complex relations than a sequence.

3.1. *Recursive neural networks*

Recursive neural networks (RecNNs) have been proposed in the literature as possible generalizations of RNNs, in which the sequential computational graph is replaced by a tree structure [32]. This means that a node hidden state still depends on the previous step in the computational graph, similarly to the RNN, but the step itself is defined by a binary tree instead of a sequence. This feature opens up the possibility of adding extra domain knowledge information into the network architecture itself when building the tree. A scheme of a recursive structure based on a binary tree is shown in Fig. 14.

When building a RecNN, entries in a sequence are combined via a learned function, with a predetermined binary tree structure. Therefore, a hidden state combining entries \mathbf{x}^i and \mathbf{x}^j is represented by $h = f(\mathbf{x}^i, \mathbf{x}^j, \theta)$, where θ represents the learnable parameters. This function $f(\cdot, \cdot, \theta)$ is then used in all binary combinations, which allows

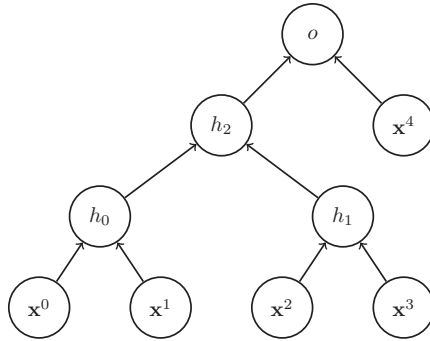


Fig. 14. Scheme of a recursive binary tree structure algorithm acting on a sequence of inputs $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4$. The h_i nodes correspond to hidden states performing the combination of two other nodes, while the o node represents the output.

the architecture to act on variable-length sequences.^d Another advantage of RecNNs over RNNs is its lower complexity, since the computations are not performed per sequence entry anymore.

Studies utilizing RecNNs with trees reproducing the jet clustering history as a basis for jet representation have been performed in the context of jet classification [33–35] and will be detailed below. They show that RecNNs are able to learn a fixed-length jet embedding from a variable length tree structure built from the jet constituents. This embedding can be further used for different tasks, such as classification and regression.

Applications to Jet Physics

Initial studies used the RecNN architecture for jet discrimination [33], using simulated events processed through a simplified detector simulation (Delphes). The signal jets are comprised of hadronically decaying W-bosons reconstructed as a single $R = 1.0$ anti- k_T -jet (boosted jet), while the background is taken from purely QCD hard scattering proton-proton collisions. The study performs a comparison

^dThis is another instance of weights sharing in neural networks, a necessity when dealing with variable length inputs.

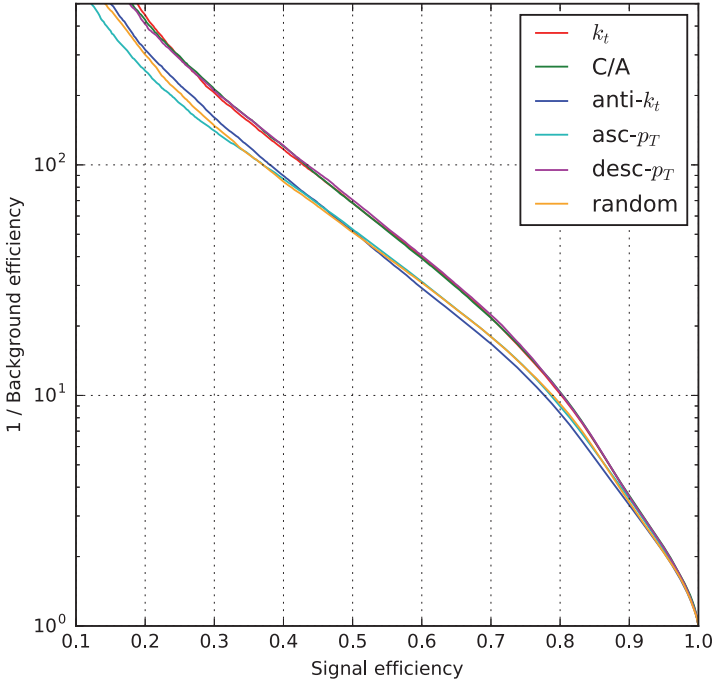


Fig. 15. Performance of RecNN-based algorithm for identifying boosted W -jets with respect to jets produced via purely QCD interactions [33]. The horizontal axis shows the efficiency of correctly identifying W -jets, while the vertical axis shows one over the efficiency of incorrectly identifying QCD jets as W -jets. The different line colors correspond to different jet clustering algorithms used to build the binary tree topology in the RecNN.

between RecNN architectures based on different binary trees, corresponding to different jet clustering algorithms histories. The result of this comparison is shown in Fig. 15. It is interesting to note that, even though the jets have been initially clustered with the anti- k_T algorithm, other clustering histories, such as k_T , perform better. This is consistent with previous observations that k_T outperforms anti- k_T in terms of identifying substructures in jets. In general, this variation in performance is an evidence for the strong dependence of RecNNs architectures on the choice of binary tree topology.

The performance is also studied when adding gating to the RecNN nodes. Similarly to LSTMs and GRUs, gated structures are used to

regulate how much information is passed through the binary tree. An improvement in performance is observed with gating, but the importance of topology is still dominant in the results.

This study also shows that the learned RecNN jet embedding can be used for event-wide discrimination, identifying beyond the Standard Model processes involving boosted W - and Z -jets from purely QCD processes. In this case, these RecNN embeddings of the individual jets in an event are passed through as a p_T -ordered sequence to a GRU-based architecture which performs the classification task. It was observed that the usage of RecNN-based embeddings improves significantly the event classification performance with respect to using a GRU-only architecture acting on sequences of jets.

A similar strategy has also been employed for quark vs. gluon jet discrimination [34], showing a slight improvement on a baseline boosted decision tree-based algorithm. Quark vs. gluon discrimination is an important avenue of work at the LHC, due to the enormous gluon background from soft hadronic interactions, which have a strong impact on analyses with light quarks in the final state. One important application of this class of algorithms is identifying the hard scatter final state light jets involved in production of the Higgs boson via vector boson fusion.

As in the previous study, simulated events are processed with Delphes and jets are clustered with the anti- k_T algorithm with $R = 0.7$ for high jets with $p_T > 1$ TeV and $R = 0.4$ for jets with lower p_T . Purely QCD events with two jets from the hard scatter (dijet events) are produced separately for when the hard scatter partons are gluons (background), or up, down or strange quarks (signal). The discrimination achieved with the RecNN under different jet reconstruction strategies is shown in Fig. 16, together with a baseline approach based on a BDT with engineered features (jet shape and kinematics). Three types of jet reconstructions are compared: using calorimeter towers only (“nopflow”); using particle identification for neutral hadrons, photons, and positively and negatively charged particles, encoded in one-hot vectors for each jet constituent in the tree (“one-hot”); using a p_T weighted jet charge defined by the clustering tree (“ptwcharge”). Although little difference is observed with

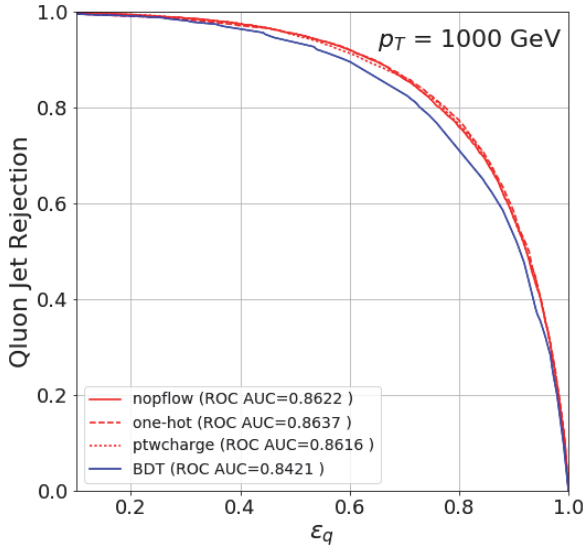


Fig. 16. Performance of RecNN-based algorithm for discriminating quark and gluon jets [34]. The horizontal axis shows the efficiency of correctly identifying quark initiated jets, while the vertical axis shows one minus the efficiency of incorrectly identifying gluon jets as quark jets. The red lines correspond to the RecNN models trained with different jet reconstruction techniques. The blue line shows the performance of a baseline BDT based on engineered features related to jet shape and kinematics.

respect to the investigated jet reconstruction schemes, a significant improvement is obtained over the BDT baseline.

Recent studies have also compared RecNNs to RNNs and CNNs when tasked to estimate jet charges [35]. In the same spirit of identifying jet flavor, identifying jet charges can help with mitigating the enormous multijet background present in hadronic LHC searches. Requiring that two jets forming a neutral resonance have opposite charges could potentially reach that goal, assuming a good charge reconstruction resolution is achieved.

For these studies, jets from up quarks were used as proxies for positively charged jets, while jets from down quarks were used for negatively charged jets. Jets were simulated from QCD hard scattering processes in proton–proton interactions, and clustered from final state particles with the anti- k_T algorithm with $R = 0.4$. To test the

performance of CNN's on jet images, the jets were formatted into $\delta\phi \times \delta\eta = 33 \times 33$ pixel box, with pixel intensities referring to the transverse momenta going into that specific pixel, and to the sum of track charges weighted by their momenta over the tracks corresponding to that pixel.

Results are shown comparing different ML-based models acting on jet constituents: standard CNNs, residual CNNs, RNNs and RecNNs. The RNN architecture implemented is based on GRUs, with LSTMs performing similarly; the jet constituents are ordered in the input sequence by their p_T , with the ordering based on distance to the jet axis performing equally. These different algorithms' performances are presented in Fig. 17 in terms of Significance Improvement Curves (SIC), defined by $\epsilon_s/\sqrt{\epsilon_b}$, where ϵ_s is the efficiency of correctly identifying down quark initiated jets, and ϵ_b is the efficiency of incorrectly identifying up quark jets as down jets. They are compared to more standard strategies based on engineered features. Overall, the algorithms acting on jet constituents consistently outperform the baseline approaches.

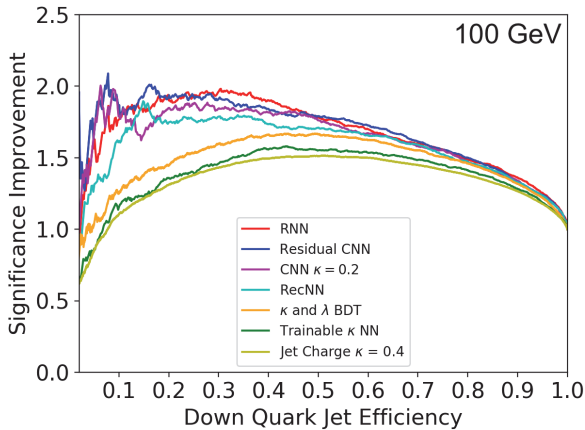


Fig. 17. Performance of ML-based algorithms acting on jet constituents to discriminate up and down jets based on their electric charges [35]. The horizontal axis shows the efficiency of correctly identifying down quark initiated jets (ϵ_s), while the vertical axis shows the ratio $\epsilon_s/\sqrt{\epsilon_b}$, where ϵ_b is the efficiency of incorrectly identifying up quark jets as down jets.

3.2. *Transformers and deep sets*

As shown by some of the studies described above, the ordering choice can be detrimental to the RNNs performance. One recent strategy proposed to overcome this ordering dependency is attention mechanisms [36]. The main working point of attention mechanisms can be understood with a simple NLP example. Within translation problems, two languages often display different semantic structures for the same sentence: “a yellow cat” in English becomes “um gato amarelo” in Portuguese, with the words for “cat” and “yellow” switching positions. This relationship is difficult to be learned through standard RNNs in which the input and output sequences have a defined order. To solve this issue, network architectures with attention will directly learn correlations between the entries in the input sequence and the entries in the output sequence, which might not be encoded in the sequences ordering.

Different strategies and architectures involving ideas related to neural network attention have been proposed to mitigate the ordering issue. In particular, transformer networks [37], which employ a self-attention technique, learning two-by-two correlations between the sequence inputs themselves through attention weights. Transformers have become common tools in NLP, with pretrained models such as BERT (bidirectional encoder representations from transformers) [38] being adopted by Google in its search engine for better understanding search queries.

Another method which aims to exploit correlations on a variable-length input structure is the deep sets architecture [39]. Deep sets are particularly suited for situations in which the ordering is not well defined, as it treats the variable-length input sequence as a permutation invariant set. A similar deep sets architecture was used [40] to learn representations for events in collider experiments (particle flow networks). These particle flow networks act on lists of particles, such as jet constituents, learns a combined representation through dense layers with shared weights, and sums them into a single, object-wide representation.

A similar architecture to the particle flow networks was used in the ATLAS experiment as an alternative to the RNN-based model

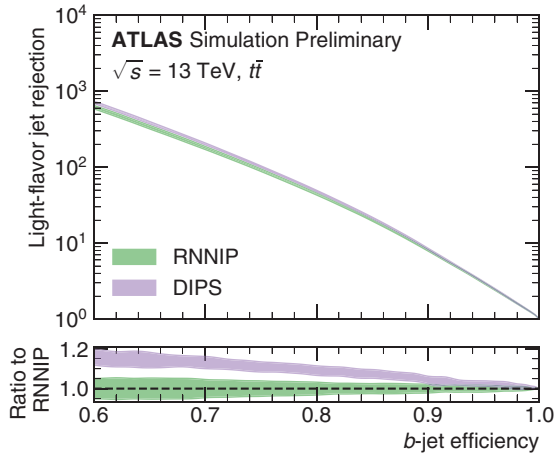


Fig. 18. Performance of the RNNIP [16] and DIPS [41] heavy flavor identification algorithms in the ATLAS experiment. The horizontal axis shows the efficiency of correctly identifying b -jets, while the vertical axis shows the inverse of the efficiency of incorrectly identifying light flavor jets as b -jets. The violet band shows the performance the DIPS performance, while the green band represents RNNIP. The width and central value of the curves represent the standard deviation and mean of the light flavor jets rejection for a given b -jet efficiency for five different network trainings. Performances were measured for jets with a transverse momentum above 20 GeV, in a simulated dataset of top quark pairs, at center-of-mass energy of 13 TeV.

in the context of heavy flavor identification [41]. The deep sets-based ATLAS heavy flavor discriminant (DIPS) outperforms its RNN analogous (RNNIP, described in Sec. 2.1) version of the RNNIP ATLAS heavy flavor discriminant by up to 20% for b -jet efficiencies larger than 60%, while using the same inputs, as seen in Fig. 18. DIPS has also been found to reduce significantly the training and evaluation time with respect to RNNIP, due to its parallelizability. The ability of parallelizing computations on each sequence element is an important feature of this model, particularly for applications in which the network evaluation time is limited.

4. Conclusion

Sequence-based machine learning algorithms have a long and rich history in the context of natural language processing. While the idea

of representing a jet as a sequence of its constituents is not new in particle physics, as evidenced by jet clustering algorithms, the usage of ML concepts exploiting this representation is relatively recent when compared to computer vision algorithms. Even so, the application of RNNs to jet physics in particular has been a fruitful avenue of research in the past few years. Special attention was given to the predictive power of these models, with the successful application of LSTM-based neural network architectures to different types of jet classification tasks. More generally, recurrent structures were shown to be well-suited to describe jet clustering histories, leading to a full probabilistic model of a jet given its constituents.

Recent work has also been focused on expanding the basic RNN ideas of hierarchical context learning to more physics inspired architectures, such as recursive neural networks. However, while these models achieve high precision when the correct choice of input structure is used — either the binary tree in a RecNN or the ordered sequence itself for RNNs — their performances can be significantly degraded given a wrong structure choice. This is particularly difficult to deal with when the input has variable sized length but the data structure is not obvious. For example, how one chooses to order the set of tracks inside the jet will depend on the task to be performed: ordering in impact parameter significance can be suited for b -jet identification, but not for quark vs. gluon discrimination. With that in mind, algorithms in which the data structure itself is either learned, such as transformers or graphs, or invariant under certain problem transformations, such as permutation invariant sets, have shown a great potential for future studies.

References

- [1] B. Hammer, On the approximation capability of recurrent neural networks, *Neurocomputing*. **31**(1) (2000) 107–123; doi: 10.1016/S0925-2312(99)00174-5.
- [2] M. Schuster and K. Paliwal, Bidirectional recurrent neural networks, *IEEE. Trans. Signal Process.* **45** (1997) 2673–2681; doi: 10.1109/78.650093.
- [3] R. Pascanu, T. Mikolov and Y. Bengio, Understanding the exploding gradient problem (2012); arXiv:1211.5063.
- [4] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.* **9** (1997) 1735–80; doi: 10.1162/neco.1997.9.8.1735.

- [5] K. Cho, B. van Merriënboer, Ç. Gülçehre, F. Bougares, H. Schwenk and Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation (2014); 1406.1078.
- [6] C. Olah, Understanding LSTM networks (2015); <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [7] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman and A. Schwartzman, Jet-images — deep learning edition, *J. High Energy Phys.* **07** (2016) 069; doi: 10.1007/JHEP07(2016)069.
- [8] M. Cacciari, G. P. Salam and G. Soyez, The anti- k_t jet clustering algorithm, *J. High Energy Phys.* **04** (2008) 063; doi: 10.1088/1126-6708/2008/04/063.
- [9] LHC Higgs Cross Section Working Group, S. Heinemeyer, C. Mariotti, G. Passarino and R. Tanaka (eds.), CERN-2013-004 (CERN, Geneva, 2013) Handbook of LHC Higgs Cross Sections: 3. Higgs Properties; doi: 10.5170/CERN-2013-004.
- [10] D. Buttazzo, G. Degrassi, P. P. Giardino, G. F. Giudice, F. Sala, A. Salvio and A. Strumia, Investigating the near-criticality of the higgs boson, *J. High Energy Phys.* **2013**(12) (2013); doi: 10.1007/jhep12(2013)089.
- [11] P. Zyla *et al.*, Review of particle physics, *Prog. Theor. Exp. Phys.* **2020**(8) (2020) 083C01; doi: 10.1093/ptep/ptaa104.
- [12] Standard Model Summary Plots Spring 2019, Technical Report, ATL-PHYS-PUB-2019-010, CERN, Geneva (2019). URL <https://cds.cern.ch/record/2668559>.
- [13] Topological b -hadron decay reconstruction and identification of b -jets with the JetFitter package in the ATLAS experiment at the LHC, Technical Report, ATL-PHYS-PUB-2018-025, CERN, Geneva (2018). URL <http://cds.cern.ch/record/2645405>.
- [14] G. Aad *et al.*, ATLAS b -jet identification performance and efficiency measurement with $t\bar{t}$ events in pp collisions at $\sqrt{s} = 13$ TeV, *Eur. Phys. J. C* **79**(11) (2019); doi: 10.1140/epjc/s10052-019-7450-8.
- [15] D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban and D. Whiteson, Jet flavor classification in high-energy physics with deep neural networks, *Phys. Rev. D* **94**(11) (2016) 112002; doi: 10.1103/PhysRevD.94.112002.
- [16] ATLAS Collaboration, Identification of jets containing b -hadrons with recurrent neural networks at the ATLAS experiment, Technical Report, ATL-PHYS-PUB-2017-003, CERN, Geneva (2017). URL <https://cds.cern.ch/record/2255226>.
- [17] A. Sirunyan *et al.*, Identification of heavy-flavour jets with the cms detector in pp collisions at 13 TeV, *J. Instrum.* **13**(05) (2018) P05011; doi: 10.1088/1748-0221/13/05/p05011.
- [18] Performance of b tagging algorithms in proton–proton collisions at 13 TeV with Phase 1 CMS detector (2018). URL <https://cds.cern.ch/record/2627468>.
- [19] A. Ali, F. Barreiro and T. Lagouri, Prospects of measuring the CKM matrix element $|V_{ts}|$ at the LHC, *Phys. Lett. B* **693**(1) (2010) 44–51; doi: 10.1016/j.physletb.2010.08.014.

- [20] J. Erdmann, A tagger for strange jets based on tracking information using long short-term memory, *J. Instrum.* **15**(01) (2020) P01021; doi: 10.1088/1748-0221/15/01/P01021.
- [21] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens and M. Selvaggi, Delphes 3: A modular framework for fast simulation of a generic collider experiment, *J. High Energy Phys.* **2014**(2) (2014); doi: 10.1007/jhep02(2014)057.
- [22] ATLAS Collaboration, Identification of hadronic tau lepton decays using neural networks in the ATLAS experiment, Technical Report, ATL-PHYS-PUB-2019-033, CERN, Geneva (2019). URL <https://cds.cern.ch/record/2688062>.
- [23] A. Sirunyan *et al.*, Identification of heavy, energetic, hadronically decaying particles using machine-learning techniques, *J. Instrum.* **15** (06) (2020) P06005; doi: 10.1088/1748-0221/15/06/p06005.
- [24] M. Wobisch and T. Wengler, Hadronization corrections to jet cross-sections in deep inelastic scattering, in *Workshop on Monte Carlo Generators for HERA Physics (Plenary Starting Meeting)* (1998), pp. 270–279.
- [25] M. Aaboud *et al.*, Performance of top-quark and W -boson tagging with ATLAS in Run 2 of the LHC, *Eur. Phys. J. C* **79**(5) (2019) 375; doi: 10.1140/epjc/s10052-019-6847-8.
- [26] S. Egan, W. Fedorko, A. Lister, J. Pearkes and C. Gay, Long short-term memory (LSTM) networks with jet constituents for boosted top tagging at the LHC (2017); arXiv:1711.09059.
- [27] J. Pearkes, W. Fedorko, A. Lister and C. Gay, Jet constituents for deep neural network based top quark tagging (2017); arXiv:1704.02124.
- [28] D. Krohn, J. Thaler and L.-T. Wang, Jet trimming, *J. High Energy Phys.* **2010**(2) (2010); doi: 10.1007/jhep02(2010)084.
- [29] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, JUNIPR: A Framework for Unsupervised Machine Learning in Particle Physics, *Eur. Phys. J. C* **79**(2) (2019) 102; doi: 10.1140/epjc/s10052-019-6607-9.
- [30] S. Catani, Y. Dokshitzer, M. Seymour and B. Webber, Longitudinally-invariant K -clustering algorithms for hadron-hadron collisions, *Nucl. Phys. B* **406**(1) (1993) 187–224; doi: 10.1016/0550-3213(93)90166-M.
- [31] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, Binary JUNIPR: an interpretable probabilistic model for discrimination, *Phys. Rev. Lett.* **123**(18) (2019) 182001; doi: 10.1103/PhysRevLett.123.182001.
- [32] L. Bottou, From machine learning to machine reasoning (2011); arXiv: 1102.1808.
- [33] G. Louppe, K. Cho, C. Becot and K. Cranmer, QCD-aware recursive neural networks for jet physics, *J. High Energy Phys.* **01** (2019) 057; doi: 10.1007/JHEP01(2019)057.
- [34] T. Cheng, Recursive neural networks in quark/gluon tagging, *Comput. Softw. Big Sci.* **2**(1) (2018) 3; doi: 10.1007/s41781-018-0007-y.
- [35] K. Fraser and M. D. Schwartz, Jet charge and machine learning, *J. High Energy Phys.* **10** (2018) 093; doi: 10.1007/JHEP10(2018)093.

- [36] D. Bahdanau, K. Cho and Y. Bengio, Neural machine translation by jointly learning to align and translate, in *3rd Int. Conf. Learning Representations, ICLR 2015*, eds. Y. Bengio and Y. LeCun, San Diego, CA, USA, May 7–9, 2015 (2015); arXiv:1409.0473.
- [37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, Attention is all you need (2011); arXiv: 1706.03762.
- [38] J. Devlin, M. Chang, K. Lee and K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in *Proc. 2019 Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*, eds. J. Burstein, C. Doran and T. Solorio, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pp. 4171–4186 (Association for Computational Linguistics, 2019), pp. 4171–4186; doi: 10.18653/v1/n19-1423.
- [39] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov and A. J. Smola, Deep sets (2017); arXiv:1703.06114.
- [40] P. T. Komiske, E. M. Metodiev and J. Thaler, Energy flow networks: deep sets for particle jets, *J. High Energy Phys.* **2019**(1) (2019); doi: 10.1007/JHEP01(2019)121.
- [41] Deep Sets based neural networks for impact parameter flavour tagging in ATLAS, Technical Report, ATL-PHYS-PUB-2020-014, CERN, Geneva (2020). URL <https://cds.cern.ch/record/2718948>.

Part VII

Physics Inference

This page intentionally left blank

Chapter 16

Simulation-Based Inference Methods for Particle Physics

Johann Brehmer* and Kyle Cranmer[†]

New York University, New York, NY 10003, USA

**jb6504@nyu.edu*

[†]kyle.cranmer@nyu.edu

Our predictions for particle physics processes are realized in a chain of complex simulators. They allow us to generate high-fidelity simulated data, but they are not well-suited for inference on the theory parameters with observed data. We explain why the likelihood function of high-dimensional LHC data cannot be explicitly evaluated, why this matters for data analysis, and reframe what the field has traditionally done to circumvent this problem. We then review new simulation-based inference methods that let us directly analyze high-dimensional data by combining machine learning techniques and information from the simulator. Initial studies indicate that these techniques have the potential to substantially improve the precision of LHC measurements. Finally, we discuss probabilistic programming, an emerging paradigm that lets us extend inference to the latent process of the simulator.

1. Particle Physics Measurements as a Simulation-Based Inference Problem

1.1. *A fundamental problem for LHC measurements*

Among the sciences, particle physics has the luxury of having a very well-established theoretical basis. Quantum field theory provides a framework not only for the Standard Model, but also for theories of physics beyond the standard model (BSM). We almost take for granted the predictive power of our theories, but the way our field formulates searches for new physics in terms of hypothesis tests and

confidence intervals is critically tied to the fact that we have predictive models to test in the first place.

Often we seem to equate the predictions of a theory with Feynman diagrams and the matrix element for a hard scattering process, which in turn can be used to predict a fully differential cross-section. Of course, that is not the full story, as one must include parton density functions and quarks and gluons give rise to a parton shower and subsequent hadronization process. Moreover, we observe electronic signatures tied to scintillation, ionization, etc. in our detectors, not the final-state particles directly. Therefore, the predictive model for a theory must incorporate the response of the detector to the final state particles.

While all of these points are well known to an experimental particle physicist, it has not been customary to describe the full simulation chain explicitly as a probabilistic model for the data. Why is that? In part that is because we have no explicit closed-form equation to write down nor do we have a function that we can provide to `Minuit` [1] that describes the probability distribution for the raw data in terms of parameters that appear in the Lagrangian for a given theory. Nevertheless, we can produce synthetic data using Monte Carlo simulation tools like `MadGraph` [2], `Sherpa` [3], `Pythia` [4], `Herwig` [5], and `GEANT4` [6].

The colloquial term or jargon for both the simulation tools and the synthetic data they produce is *Monte Carlo*. This term refers to methods that sample from probability distributions to compute an integral. Particle physics simulators use such methods to compute the cross-section of a process by sampling a number of events following the probability distribution

$$p(x, z_d, z_s, z_p | \theta) = p_x(x | z_d) p_d(z_d | z_s) p_s(z_s | z_p) p(z_p | \theta). \quad (1)$$

Here the four probability densities describe the sequence of hard process, parton shower, interactions with the detector, and construction of observables. This factorization represents the reality that the `Geant4` detector simulation does not depend on the Lagrangian parameters that are modified in the hard scattering modeled by `MadGraph`, and so on. The vector z_p is the parton-level phase-space point of a simulated event (i.e. the parton four-momenta, helicities,

and charges); the vector z_s summarizes the parton shower simulation, including the stable particles that emerge from it; z_d are the interactions in the detector. These three vectors collectively define the “Monte-Carlo truth record” of a simulated event and are the *latent variables* of the process: we cannot measure them, and in fact they are only well defined within a given simulator code. Finally, x is the vector of observables. While a real-life observation consists of tens of millions of sensor read-outs, one can consider the reconstruction of the event as part of the measurement process and take x as a vector of four-momenta and other properties of the reconstructed particles. In Table 1 we provide a look-up table for these and other symbols that appear in this chapter and translate between particle physics and machine learning or statistics nomenclature.

There is an established chain of high-fidelity simulators that can sample events from the probability density in Eq. (1). However, statistical inference — quantifying the degree to which parameter values θ are in agreement with an observed set of events $\mathcal{D} = \{x_i\}_{i=1}^n$ — is surprisingly challenging. Why? The key quantity for both frequentist and Bayesian inference method is the likelihood function $p_{\text{full}}(\mathcal{D}|\theta)$, the probability density of an observed set of events \mathcal{D} as a function of the parameters θ . The full likelihood function is given by

$$p_{\text{full}}(\mathcal{D}|\theta) = \text{Pois}(n|\epsilon L \sigma(\theta)) \prod_i p(x_i|\theta), \quad (2)$$

where $\text{Pois}(n|\epsilon L \sigma(\theta))$ is the Poisson probability density for n observed events, efficiency and acceptance factors ϵ , integrated luminosity L , total cross-section $\sigma(\theta)$, and where

$$p(x|\theta) = \int dz_d \int dz_s \int dz_p p(x, z_d, z_s, z_p|\theta) \quad (3)$$

is the probability density for an individual event to have data x . This likelihood function involves integrals over the entire parton-level phase space, all possible shower histories, and all possible detector interactions compatible with the measurement x . The integral over this enormous space clearly cannot be computed in practice, so we cannot directly evaluate the likelihood of an observed event under different parameter values θ . This means that we cannot directly find

Table 1. Dictionary of symbols that appear in this chapter (derived from [7]).

Symbol	Meaning	ML abstraction
θ	Theory parameters	Parameters of interest
x	All observables	Features
v	1–2 selected kinematic variables	Summary statistics
z_p	Parton-level four-momenta	Latent variables
z_s	Parton shower history	Latent variables
z_d	Detector interactions	Latent variables
$z = (z_p, z_s, z_d)$	Full simulation history of event	All latent variables
$p_{\text{full}}(\{x\} \theta)$	Full likelihood function, see Eq. (2)	Implicit density
$p(x \theta)$	Kinematic likelihood for single event (normalized fully differential cross-section, Eq. (3))	Implicit density
$p_p(z_p \theta)$	Parton-level distribution	Tractable density
$p_s(z_s z_p)$	Parton-shower effects	Implicit density
$p_d(z_s z_p)$	Detector effects	Implicit density
$p_x(x z_d)$	Detector readout	Implicit density
$r(x \theta)$	Likelihood ratio function, see Eq. (4)	
$r(x, z \theta)$	Joint likelihood ratio, see Eq. (8)	Unbiased est. of $r(x \theta)$
$t(x)$	Score (locally optimal obs., Eq. (10))	
$t(x, z \theta)$	Joint score, see Eq. (9)	Unbiased est. of score
$\hat{\theta}$	Best fit for theory parameters	Estimator for θ
$\hat{p}(x \theta)$	Parameterized estimator for likelihood	
$\hat{r}(x \theta)$	Parameterized estimator for likelihood ratio	
$\hat{s}(x \theta)$	Parameterized classifier decision function	
$\hat{t}(x)$	Estimator for score	
$\hat{p}_{tf}(x z_p)$	Approximate shower and detector effects (transfer function)	

the maximum-likelihood estimators that best fit a given observation, construct confidence limits based on a likelihood ratio test statistic, or compute the Bayesian posterior $p(\theta|x)$!

The task of performing statistical inference when the data generating process does not have a tractable likelihood is known as *simulation-based* or *likelihood-free inference*. This case is not at all unique to particle physics. The formulation of this problem in a

common, abstract language has led to statisticians, computer scientists, and domain scientists from various fields developing powerful methods for simulation-based inference together. While this chapter focuses on the particle physics case, the methods apply equally to a range of problems for instance in neuroscience, cosmology, or epidemiology.

1.2. *Solving the problem with summary statistics*

If the intractability of the likelihood function is such a problem, how have high-energy physicists successfully analyzed particle collisions for decades? The reason that this problem is rarely acknowledged explicitly is that particle physicists have a track record of developing a good intuition about processes they study and finding powerful summary statistics for them. Summary statistics are individual variables that condense a high-dimensional observation. Typical examples are the reconstructed mass of a decaying unstable particle, decay angles between decay products, or other kinematic variables [8, 9]. An ideal summary statistics vector v captures all of the relevant information in the observed event x relevant to the parameter θ , while being of much lower dimensionality. Given one or two summary statistics, we can easily compute the likelihood function $p(v|\theta)$ with histograms, kernel density estimation, or other density estimation techniques and then find the maximum-likelihood estimator in the parameter space and construct confidence limits based on the (profile) likelihood ratio test statistic [10, 11]. This approach has been the workhorse of statistical analysis in collider physics for decades.

Note that most uses of machine learning in experimental particle physics take place within this approach. Experimental particle physicists have embraced the use of multivariate models (commonly boosted decision trees or fully connected neural networks) in the event selection, as reviewed in Chapter 2. The statistical analysis of the events that pass this selection is then still based on histograms of kinematics-based summary statistics or the neural network output itself.

The reduction of data to summary statistics also enables approximate Bayesian computation (ABC) [12, 13], a simulation-based

inference method that is gaining popularity in cosmology and is widely used in many scientific fields outside of physics. It directly targets Bayesian inference, using repeated runs of the simulator together with an accept–reject criterion to draw parameter samples that approximately follow the posterior.

Both the histogram method popular in particle physics and ABC suffer from the curse of dimensionality: the number of simulations required scales exponentially with the dimensionality of x or v . This is why they only work with a low-dimensional statistic v and cannot be effectively applied to high-dimensional data x . However, finding suitable summary statistics is a difficult and task-dependent problem and almost any choice of summary statistics discards some information. As a result, data analysis based on summary statistics typically leads to reduced sensitivity and statistical power.

1.3. *The frontier of simulation-based inference*

In the next sections, we will describe modern simulation-based inference methods that allow us to analyze higher-dimensional data, improve the quality of inference, and improve the sample efficiency. Three developments are the key drivers behind these improvements [14]:

- (1) The revolution in machine learning provides us with powerful surrogate models for the likelihood, likelihood ratio, or posterior function, or for optimal summary statistics. We can thus tap into the ability of modern machine learning methods to learn useful representations directly from high-dimensional data.
- (2) Active learning methods iteratively use past results to steer the next simulations, leading to a better sample efficiency.
- (3) Integrating inference capabilities with the simulation code and augmenting the training data with additional information that can be extracted from the simulator can substantially improve sample efficiency and quality of inference.

Against the backdrop of these three broad trends, many different inference algorithms have been proposed in recent years, see [14] for

an overview. Here we focus on a few methods that are particularly relevant for particle physics. In Sec. 2, we discuss techniques that aim to estimate the likelihood function or the likelihood ratio function, ranging from the Matrix-Element Method to machine learning-based methods to techniques that bring together matrix-element information and machine learning. Section 3 covers methods that aim to define powerful summary statistics, from parton-level Optimal Observables to neural network surrogates for the score function. We summarize and compare the main methods we discuss in Table 2. In the following sections, we will briefly discuss diagnostic tools and systematic uncertainties as well as software implementations of these ideas. In Sec. 5, we focus more on the latent process of the simulators and describe the paradigm of probabilistic programming. We discuss implementations of these methods in the HEP software stack in Sec. 6, before concluding with a summary in Sec. 7.

Table 2. Simulation-based inference methods for particle physics. We classify methods by the key quantity that is estimated in the different approaches, by whether they rely on a manual choice of summary statistics, are based on a transfer-function approximation (“TF”), whether their optimality depends on a local approximation (“local”), by whether they use any other functional approximations such as a histogram binning or a neural network (“NN”), whether they leverage matrix-element information (“ $|\mathcal{M}|^2$ ”), and by the computational evaluation cost. Derived from a table in [15].

Method	Estimates	Approximations				$ \mathcal{M} ^2$	Comp. cost
		Summaries	TF	Local	Functional		
Histograms	$\hat{p}(v \theta)$	✓			binning		curse of dim.
ABC	$\theta \sim p(\theta v)$	✓			ϵ -kernel		curse of dim.
MEM	$\hat{p}(x \theta)$		✓		integral	✓	high (TF)
NDE	$\hat{p}(x \theta)$				NN		amortized
SCANDAL	$\hat{p}(x \theta)$				NN	✓	amortized
CARL	$\hat{r}(x \theta)$				NN		amortized
RASCAL etc	$\hat{r}(x \theta)$				NN	✓	amortized
OO	$\hat{t}(x)$		✓	✓	integral	✓	high (TF)
SALLY	$\hat{t}(x)$			✓	NN	✓	amortized

2. Inference with Surrogates

The first class of methods that we discuss tackles the problem head-on and constructs an estimator for the likelihood function $p(x|\theta)$ or the closely related likelihood ratio function

$$r(x|\theta) = \frac{p(x|\theta)}{p_{\text{ref}}(x)}, \quad (4)$$

where the denominator is some reference distribution, for instance using a reference value of the parameter points such as the Standard Model, a model average of multiple parameter points, or uniform phase space.

Once we have such an estimator, which we will denote $\hat{p}(x|\theta)$ or $\hat{r}(x|\theta)$, we can immediately use it in the established statistical pipeline: we can find the maximum-likelihood estimator for instance as

$$\hat{\theta}_{\text{MLE}} = \arg \max_{\theta} \text{Pois}(n|L\sigma(\theta)) \prod_i \hat{r}(x_i|\theta) \quad (5)$$

and similarly construct exclusion limits based on asymptotic properties of the (profile) likelihood ratio [16]. Additionally, we can use the resulting likelihood ratio test statistic together with toy Monte Carlo to guarantee coverage, as discussed in Sec. 4.

2.1. *An approximation: The matrix-element method*

The matrix-element method (MEM) [17–32] approximates the likelihood in Eq. (3) by replacing the precise model of the effects of shower and detector with a simple, tractable transfer function $\hat{p}_{tf}(x|z_p)$. This simplifies the marginal distribution that would involve integrating over a large number of microscopic interactions to a convenient probability density such as a Gaussian. The MEM likelihood is given, schematically, by

$$\begin{aligned} \hat{p}_{MEM}(x|\theta) &= \int dz_p \hat{p}_{tf}(x|z_p) p(z_p|\theta) \\ &\sim \frac{1}{\sigma(\theta)} \int dz_p \hat{p}_{tf}(x|z_p) |\mathcal{M}(z_p|\theta)|^2, \end{aligned} \quad (6)$$

where $|\mathcal{M}(z_p|\theta)|^2$ is the squared matrix element evaluated at a phase-space point z_p and parameters θ and for simplicity we have left out parton densities as well as phase-space and efficiency factors. Since the integrand is tractable and the integral is over a much lower-dimensional space than the one in Eq. (3), it is feasible — though expensive — to compute this approximate likelihood function.^a

In some processes, particularly those involving only leptons and photons, the MEM can give a reliable estimate of the true likelihood. However, jets are less well modeled by transfer functions and additional jet radiation is difficult to describe in this approach (with challenges related to ME/PS matching, higher orders, etc.). Finally, the MEM still requires a computationally expensive numerical integration *for every event that is evaluated*, which can be prohibitive.

2.2. Learning the likelihood

Rather than computing the integral in the likelihood for every event to be evaluated, we can fit a surrogate model to data from the simulator and then use that for inference. Such a surrogate model needs to be flexible enough to accurately represent a complicated and multimodal probability distribution, we have to fit it to limited training data, and its likelihood function needs to be computed efficiently. Kernel density estimation has been used in this context [37], but it was limited to roughly five-dimensional data. Recently, several machine learning models have been developed for this task, which are effective for estimating distributions of high-dimensional data. In particular, normalizing flows [38, 39] transform a random variable with a tractable probability density through multiple learnable bijections so that the distribution of the output is given by the change-of-variable formula. This defines a flexible probabilistic model with a tractable likelihood function.

^aThis approach has also been extended to include an explicit calculation of leading parton-shower effects [33–36].

Thus, let's us solve the problem of simulation-based inference in three phases [40]:

- (1) We run the usual simulator chain a number of times with different input parameters θ and saving θ together with the simulated events $x \sim p(x|\theta)$.
- (2) Next, a neural density estimator is trained to learn the conditional probability density $p(x|\theta)$. We use a single model for the whole parameter space (as opposed to individual models for a number of points along a grid in the parameter space), the parameter point θ to be evaluated is an additional input to the model. Such a *parameterized* model [41, 42] can leverage the smooth dependence on the parameter space, the probability density at each parameter point can “borrow” information from nearby points (see also Chapter 3).
- (3) After training, we can evaluate this model for arbitrary observations x and parameter points θ and efficiently get an estimator for the likelihood function $\hat{p}(x|\theta)$. We can then use this to define best-fit points and exclusion limits with the usual statistical tools.

Two aspects of this approach are noteworthy. First, we can use any state-of-the-art simulator in this approach, including shower and detector effects. Unlike in the MEM, there is no need for any approximations on the underlying physics. Second, the approach is *amortized*: after an upfront simulation and training phase, we can evaluate the approximate likelihood function very efficiently for a large number of events and parameter values.

Neural density estimators like normalizing flows have other useful properties. They are generative models, i.e. one can sample from the probability distributions they have learned. This is not only a convenient cross check, but can also be used for event generation (see Part III), to unfold reconstruction-level variables to the parton level [43], and for anomaly detection [44] (see Chapter 4).

2.3. Learning the likelihood ratio

Training a surrogate for the likelihood function actually solves a harder problem than necessary for inference. To find the maximum-likelihood parameter point and to construct exclusion limits we do not actually need to know the likelihood function itself — the likelihood *ratio* $r(x|\theta)$ defined in Eq. (4) is in fact just as useful! As it turns out, training a neural network to learn the likelihood ratio is often easier than learning the likelihood function.

The key idea is known as the *likelihood ratio trick*: a binary classifier trained to discriminate samples $x \sim p(x|\theta)$ from samples $x \sim p_{\text{ref}}(x)$ will eventually^b converge to the output $\hat{s}(x|\theta) \rightarrow p_{\text{ref}}(x)/[p(x|\theta) + p_{\text{ref}}(x)]$, which is a monotonic function of the likelihood ratio $r(x|\theta)$. In other words, we can transform the output of a classifier $\hat{s}(x|\theta)$ into an estimator for the likelihood ratio function as

$$\hat{r}(x|\theta) = \frac{1 - \hat{s}(x|\theta)}{\hat{s}(x|\theta)}. \quad (7)$$

We can use this in an inference algorithm similar to the one discussed in the previous section [41, 45–54]:

- (1) We again start by running the simulator chain, generating one set of events from a reference distribution $p_{\text{ref}}(x)$ (e.g. the SM) and a second set of events from various parameter points θ .
- (2) Next, a neural classifier is trained to discriminate between these two sets, using the binary cross-entropy as a loss function. Like before, the classifier is parameterized: the parameters θ are used as explicit inputs into the classifier.
- (3) After training, we can transform the output of the classifier into an estimator for the likelihood ratio function with Eq. (7) and an optional calibration procedure. This surrogate model can then be used to find the best-fit point and exclusion contours using established statistical tools.

^bIn the limit of a sufficiently expressive model, infinite training data, and efficient minimization of the loss function.

This method is known as CARL [41, 48, 55]. It again supports arbitrary simulators without requiring approximations on the underlying physics and is amortized, allowing for an efficient evaluation after an upfront simulation and training cost. Compared to learning the likelihood function with a neural density estimator, the CARL approach can be more sample efficient (saving computation time).

While a surrogate model for the likelihood ratio does not allow us to generate samples, it can be used for reweighting. For the simulation-based inference problem, this can be useful as a diagnostic tool. In other contexts, this ability can be used to reweight events [47, 54, 55], tune shower and detector-simulation parameters to data [54], for unfolding [56], and for anomaly detection [57], see Chapter 4.

2.4. Integration and augmentation

Both inference techniques described above — training a neural density estimator to learn the likelihood function and training a classifier to learn the likelihood ratio — treat the simulator chain as a black box that takes parameters θ as input and outputs samples $x \sim p(x|\theta)$. In reality, though, we know more about the particle physics processes. They consist of the separate pieces of parton-level generator, parton shower, and detector simulation, as given by Eq. (3); typically only the parton-level step explicitly depends on the theory parameters of interest θ .

We can leverage this understanding of the simulated process to extract more information from the simulator and use it to augment the training data for the likelihood or likelihood ratio model.^c In particular, we can access the latent variables (or Monte-Carlo truth record) $z = (z_p, z_s, z_d)$, while tools like `MadGraph` let us compute matrix elements for arbitrary theory parameters. For each simulated

^cThe extraction of the joint likelihood ratio and score is in fact more general and can be realized for many simulators [58, 59]. However, the particular structure of particle physics processes (especially, the factorization of the joint likelihood in Eq. (1)) makes it easy to compute these quantities, which in this case are closely linked to the squared matrix element.

event we can thus compute two useful quantities: the *joint likelihood ratio* [7, 58, 60]

$$\begin{aligned}
 r(x, z|\theta) &\equiv \frac{p(x, z|\theta)}{p_{\text{ref}}(x, z)} \\
 &= \frac{p_x(x|z_d) p_d(z_d|z_s) p_s(z_s|z_p) p_p(z_p|\theta)}{p_x(x|z_d) p_d(z_d|z_s) p_s(z_s|z_p) p_{p,\text{ref}}(z_p)} \\
 &= \frac{|\mathcal{M}|^2(z_p|\theta)}{|\mathcal{M}|_{\text{ref}}^2(z_p)} \frac{\sigma_{\text{ref}}}{\sigma(\theta)}
 \end{aligned} \tag{8}$$

and the joint score

$$\begin{aligned}
 t(x, z|\theta) &\equiv \nabla_{\theta} \log p(x, z|\theta) \\
 &= \frac{p_x(x|z_d) p_d(z_d|z_s) p_s(z_s|z_p) \nabla_{\theta} p_p(z_p|\theta)}{p_x(x|z_d) p_d(z_d|z_s) p_s(z_s|z_p) p_p(z_p|\theta)} \\
 &= \frac{\nabla_{\theta} |\mathcal{M}|^2(z_p|\theta)}{|\mathcal{M}|^2(z_p|\theta)} - \frac{\nabla_{\theta} \sigma(\theta)}{\sigma(\theta)}.
 \end{aligned} \tag{9}$$

Here $|\mathcal{M}|^2(z_p|\theta)$ and $|\mathcal{M}|_{\text{ref}}^2(z_p)$ are the squared matrix elements for parton-level phase space points z_p for theory parameters θ and under the reference distribution, respectively, while $\sigma(\theta)$ and σ_{ref} are the total cross-sections. The joint likelihood ratio and joint score quantify how the probability of one simulated event — fixing all of the latent variables in the simulation chain — changes if we change the theory parameters θ .

How are these two quantities useful, especially given that they depend on latent variables z that are only meaningful for simulated events, but not for real measurements? It turns out that the joint likelihood ratio $r(x, z|\theta)$ is an unbiased estimator of the likelihood ratio $r(x|\theta)$ and the joint score provides unbiased gradient information. This means that we can augment our training data with these numbers and use them as labels in a supervised learning setup, as illustrated in Fig. 2. This idea is realized in a few different algorithms, which differ by the target and loss functions they use. In the ROLR method [7, 58, 60] we directly regress on the likelihood ratio by minimizing the squared error between the neural network output and the joint likelihood ratio. The RASCAL [7, 58, 60] and

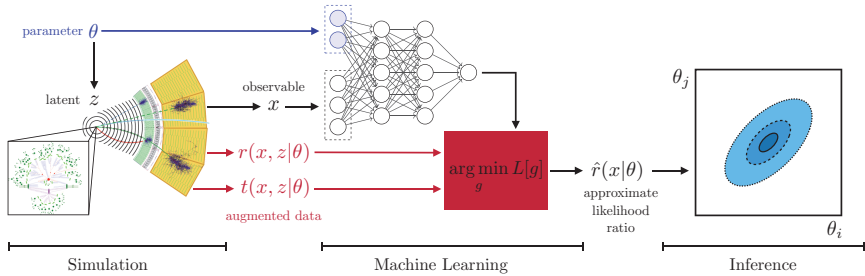


Fig. 1. Illustration of a complete inference method using the RASCAL or ALICES methods to train a surrogate for the likelihood ratio function. Figure taken from [60].

ALICES [61] techniques further improve the training of neural surrogates of the likelihood ratio function by incorporating the joint score. The SCANDAL method similarly uses the joint score to train neural surrogates for the likelihood function [58] more efficiently. After training the surrogates for the likelihood or likelihood ratio, we are again left with a neural network that can be evaluated for arbitrary events and parameter points and allows for amortized inference as before. The full workflow is schematically shown in Fig. 1.

Extracting the joint likelihood ratio and joint score during the simulation stage and augmenting the training data with it adds multiple orthogonal pieces of information to the training, as we illustrate in Fig. 2. In practice, this substantially reduces the number of simulated events that are necessary for a good performance—in some cases by multiple orders of magnitude [7, 60]!

Some particle physics measurements have even more additional structure, for instance when we are trying to constrain the Wilson coefficients of an effective field theory and the squared matrix element is a polynomial of these parameters. Incorporating this additional structure in the inference workflow can improve the efficiency even further [7, 62, 63].

2.5. Active learning

Active learning here describes a sequential workflow that alternates simulation and inference stages. The theory parameters for which

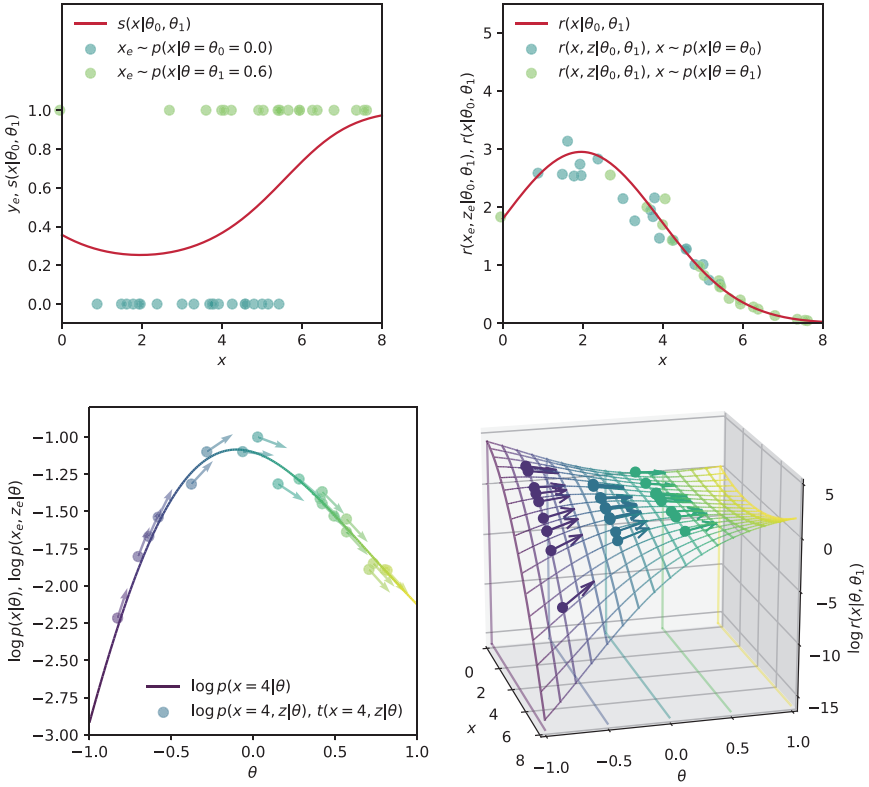


Fig. 2. Illustration of different approaches to train surrogates for the likelihood ratio function and the role of joint likelihood ratio and joint score. Figures taken from [7]. **Top left:** In the likelihood ratio trick and the CARL inference method, a classifier decision function (red) has to be learned from binary labels that are zero or one (green dots). **Top right:** The joint likelihood ratio provides noisy, but unbiased labels (green) for the likelihood ratio function to be learned (red). **Bottom left:** The joint score adds noisy, but unbiased gradient information (arrows). **Bottom right:** The RASCAL and ALICES methods combine three orthogonal pieces of information (dots with arrows), allowing a neural network to learn the likelihood ratio function (surface) more efficiently.

more events are generated are chosen such that they are expected to be most useful based on the observed data and the results of past iterations. Different algorithms have been proposed, some of which are based on neural surrogates for the likelihood [64, 65], while others target the likelihood ratio [53, 66]. While active learning is

often phrased in a Bayesian framework, these methods can be applied equally well to frequentist inference [67–69].

Active learning maximizes sample efficiency for a particular observed dataset. This is somewhat at odds with the goal of amortization, which aims to train a surrogate model that works well for multiple different datasets. While active learning can be very powerful in cases with few observed data points, it is less crucial in particle physics use cases with a large number of expected or observed events.

3. Inference with Sufficient Summary Statistics

The methods discussed in the previous section tackle simulation-based inference by learning the likelihood or likelihood ratio function in the high-dimensional data space. While these methods are powerful, they require an analysis workflow that is substantially different from the high-energy physics standards. This makes modifications to the usual software pipeline, careful cross checks, and some changes to the way that systematic uncertainties are handled necessary (more on this later).

A more incremental change to the current analysis workflow is to construct powerful summary statistics in a systematic way. After the high-dimensional data x for an event is compressed to one or a few of these summary statistics, it can be analyzed in the usual, histogram-centric way described in Sec. 1.2. The analysis workflow remains largely unchanged, except that instead of kinematic variables (like the transverse momentum of a jet) more complicated variables (like the output of a neural network) are analyzed. No essential modifications to the software pipeline or the treatment of systematic uncertainties are necessary in this approach.

So how do we find these optimal summary statistics? There are two broad strategies. The first is to try to learn a summary statistic as an intermediate step in the end-to-end analysis of the data where the objective function is, for instance, an expected significance or expected limit as in INFERNO [70] or neos [71]. This is connected to the recent discussions around differentiable programming and is discussed in Chapter 17. Optimizing an experiment-level objective

is computationally expensive, and not actually necessary since the data are independent and the likelihood factorizes as in Eq. (2).^d

Alternatively, we can look for sufficient statistics that allow us to approximate the per-event likelihood, and there are many advantages to casting the learning problem in terms of individual events. While our exposition will focus on the parameters of interest, one can consider θ to also include nuisance parameters, and profiling the nuisance parameters would then happen down-stream in the statistical inference pipeline and after the amortized learning stage described below.

The key to learning optimal observables is to consider a local approximation of the likelihood function in the parameter space. In other words, assume that we are studying parameters θ that are close to some chosen reference parameter point θ_{ref} (imagine this, for instance, to be the Standard Model). Then one can show [60, 73, 74] that the most powerful observable for measuring θ is the *score*

$$t(x) = \nabla_{\theta} \log p(x|\theta)|_{\theta_{\text{ref}}}. \quad (10)$$

This gradient vector contains one component per parameter of interest. In the neighborhood of θ_{ref} , the score components are the sufficient statistics: analyzing just $t(x)$ will yield just as much information about θ than analyzing the high-dimensional data x . By using the score as summary statistics, we are therefore not throwing away any information, at least as long we focus on parameters close to θ_{ref} . Further away from the reference point, the score components may no longer be sufficient and a histogram-based analysis will no longer be optimal.

^dIf we knew the full likelihood $p(\mathcal{D}|\theta, \nu)$ in Eq. (2), where θ are parameters of interest and ν are nuisance parameters, the final test statistic we would target would be the profile likelihood ratio $\lambda(\theta) = p(\mathcal{D}|\theta, \hat{\nu})/p(\mathcal{D}|\hat{\theta}, \hat{\nu})$, where $\hat{\theta}$ and $\hat{\nu}$ are the maximum likelihood estimator (MLE) and $\hat{\nu}$ is the conditional maximum likelihood estimator (CMLE) [72]. The numerator and denominator of the likelihood of the likelihood ratio factorize across experiments, but the values for the MLE and CMLE couple all of the events in the dataset \mathcal{D} . However, this coupling of events through the MLE and CMLE can be postponed and based on a learned surrogate for the per-event likelihood or likelihood ratio function as discussed in the previous section.

Unfortunately, like the likelihood function itself, the score is in general intractable. In the following, we will present two methods that allow us to estimate it.

3.1. *An approximation: Parton-level optimal observables*

Remember that the matrix-element method approximated the likelihood function by summarizing the effect of shower and detector with a transfer function. Parton-level optimal observables (OO) [75–77] use the same approximation to compute the score:

$$\begin{aligned}\hat{t}_{OO}(x) &= \nabla_{\theta} \log \left(\int dz_p \hat{p}_{tf}(x|z_p) p(z_p|\theta) \right) \Big|_{\theta=\theta_{\text{ref}}} \\ &= \frac{\int dz_p \hat{p}_{tf}(x|z_p) \nabla_{\theta} p(z_p|\theta_{\text{ref}})}{\int dz_p \hat{p}_{tf}(x|z_p) p(z_p|\theta_{\text{ref}})}.\end{aligned}\tag{11}$$

In practice, this method is usually applied to processes with easily identifiable final-state particles like leptons and photons. In that case, the reconstructed particle properties are simply identified with the parton-level four-momenta, $\hat{p}(x|z_p) = \prod_i \delta^4(x_i - z_{pi})$.

While this approach elegantly uses our knowledge of matrix elements, it requires substantial approximations to the underlying process, and taking into account shower or detector effects in the observable detection leads to a large computational cost for each analyzed event.

3.2. *Learning the score*

The SALLY method [7, 58, 60] trains a neural network to learn the (intractable) score function $t(x)$ including the full detector simulation. As in the methods discussed in Sec. 2.4, the first step is running the simulator chain a number of times, now using the reference parameter point θ_{ref} as input. In addition to the observation x , the joint score $t(x, z|\theta_{\text{ref}})$ defined in Eq. (9) is computed and stored for

every simulated event. In a next step, a machine learning model like a neural network $\hat{t}(x)$ is trained to minimize the mean squared error $|\hat{t}(x) - t(x, z)|^2$. It can be shown that the neural network will ultimately converge to the score function given in Eq. (10). After training, the neural network thus defines the locally most powerful observables for the measurement of θ and can be used in a standard analysis pipeline.

In addition to defining locally optimal observables, neural score estimators can also be used to compute the Fisher information, a versatile tool for sensitivity forecasting, cut optimization, and feature selection [78–80].

4. Diagnostics, Calibration, and Systematic Uncertainties

The analysis methods described in the previous sections contain some parts, in particular neural networks, that are not always easy to interpret and can be harder to debug than a standard analysis based on histograms of traditional observables. It is important to make sure that we can trust the results and quantify any systematic uncertainties. This is very similar to basing the downstream statistical analysis on histograms of neural network outputs.

Mainly we have to correctly diagnose model misspecification. Inference is always performed within the context of a statistical model, but if that model is not correct for a task at hand, the inference results will be meaningless or misleading. In the simulation-based inference methods we discuss, two types of models appear, both of which are prone to misspecification: the simulator itself and machine learning surrogates. This is similar to the distinction between the full simulation and the use of an analytic function (surrogate) to model a smooth $m_{\gamma\gamma}$ spectrum in a $H \rightarrow \gamma\gamma$ analysis.

Misspecification of the simulator occurs when **MadGraph**, **Pythia**, **Geant4**, etc. do not model the physics of LHC collisions accurately enough. This problem also plagues classical histogram-based analyses, but may be easier to diagnose and calibrate when only a single variable is studied than in the multivariate analysis methods

described here [81]. It is usually addressed by varying the parameters of the simulator, which introduces nuisance parameters α with unknown true values, and profiling over them in the statistical analysis. We can also use ideas from domain adaptation and algorithmic fairness to make the neural network less sensitive to variations in the nuisance parameters [70, 82, 83]. If possible, however, it is conceptually cleaner to explicitly include the effect of nuisance parameters in the likelihood model $\hat{p}(x|\theta, \alpha)$ or $\hat{r}(x|\theta, \alpha)$ and to use well-defined and established statistical procedures like profiling to take them into account in the downstream statistical analysis. The treatment of nuisance parameters is discussed in more detail in Chapter 17.

Misspecification of the surrogate model occurs when the neural network does not approximate the true likelihood or likelihood ratio perfectly. This is analogous to a falling exponential for the $m_{\gamma\gamma}$ spectrum not fitting the simulated data perfectly. Typical reasons are the limited number of training samples, insufficient network capacity, or an inefficient minimization of the loss function. A common issue is that the classifier $\hat{s}(x|\theta)$ will be roughly one-to-one with the true likelihood ratio, but not exactly. This can be fixed with the calibration procedure used in CARL and described in [41]. One can protect against more severe deficiencies by calibrating the inference results with toy simulations from the simulator: for every parameter point, we can run the simulator to construct the distribution of the likelihood or likelihood ratio. Ultimately this leads to confidence sets with a coverage guarantee (assuming the simulator is accurate) as in the Neyman Construction, i.e. that will never overly optimistic [7, 41]. This toy Monte Carlo approach can require a large number of simulations, especially for high-dimensional parameter spaces. For an in-depth discussion of calibration and the Neyman construction, see [7].

There are other, less computationally expensive tools to diagnose misspecification of the surrogate model. These include off-the-shelf uncertainty quantification methods for neural networks such as ensemble methods and Bayesian neural networks (see Chapter 18). In addition, one can train classifiers to distinguish data from the surrogate model and the true simulator [41], check certain expectation values of estimators of the likelihood, likelihood ratio, or score

against a known true value [7], vary unphysical reference distributions that should leave the inference result invariant [41], and compare the distribution of network outputs against known asymptotic properties [72, 84, 85]. Passing these closure tests does not guarantee that a model is correct, but failing them is an indication of an issue.

5. Probabilistic Programming

We will now switch gears and review probabilistic programming, a set of methods that are related to, but different from the simulation-based inference techniques discussed in the previous sections. Computer programs that involve random numbers and do not have deterministic input-output relationships can be thought of as specifying a probability distribution $p(\text{output}|\text{input})$. It is natural to think of simulators in this way, where the parameters of the simulator are identified with θ and the output of the simulator is identified with x . Furthermore, the values of the random variables and the other intermediate quantities inside the computer program can be thought of as latent variables z . The structure of the space of latent variables can also be complex. Consider the simple example in Fig. 3, where the list of latent variables is either $(z1, z2t)$ or $(z1, z2f, z3f)$ and depends on the control flow of the program.

It can be useful to think of the latent space of such a program as the space of its stack traces along with the values of all the variables. Take a moment to think about the complexity of the typical simulation chain going from matrix elements to parton shower and

```
def stochastic_function():
    z1 = rand()
    if z1 < 0.5:
        z2t = rand()
        x = z1 + z2t
    else:
        z2f = rand()
        z3f = rand()
        x = z1 + z2f + z3f
    return x
```

Fig. 3. Illustrative example of a stochastic function.

hadronization through the detector simulation. These programs have enormous, highly structured latent spaces. The probability that the program returns x corresponds to integrating over all the possible executions of the program that could return x ; as we argued in the introduction of this chapter, this is intractable for moderately complicated programs.

We saw in Sec. 2 how we can use machine learning surrogates to approximate the likelihood $p(x|\theta)$ or likelihood ratio $r(x|\theta)$, where the dependence on z has been marginalized or integrated out. One of the advantages of those approaches is that the surrogate models do not attempt to capture the complexity of the latent state or the joint distribution $p(x, z|\theta)$. But what if we also want to infer something about the latent variables that describe what is going on inside the simulator?

In HEP it is common to inspect the Monte Carlo truth record (i.e. z) for some set of events that satisfy some cuts to gain insight into why something happens. For instance, we might want to know what happened inside the simulation of $pp \rightarrow jj$ events that led to very large missing transverse energy, or why a jet faked a muon. To study this, we often filter a large set of events (simulated with a particular parameter setting θ), filter those events that satisfy the cuts, and then look at histograms of some particular Monte Carlo truth quantities $f(z)$ (for example, to inspect if there was a semi-leptonic b -decay or punch-through in the calorimeter). That familiar procedure is approximating the posterior distribution of $f(z)$ given that the event generated with parameter θ passes the cuts, which we can write symbolically as $p(f|\text{cuts}(x) = \text{True}, \theta)$. Similarly, the unfiltered sample can be thought of as samples from the prior $p(f|\theta)$.

The problem with the traditional approach is that the filter efficiency can be very low, and very few of the prior samples may survive to estimate the posterior. This is similar to the inefficiency found in approximate Bayesian computation, which asks for the simulator to generate a simulated x close to the observed x_{obs} . This motivates an additional language construct that allows for conditioning on random variables, which characterizes probabilistic programming. Probabilistic programming languages (PPLs) extend general-purpose

programming languages with constructs to do sampling and conditioning of random variables [86, 87].^e

The additional language constructs express the concept of sampling and conditioning, but they do not necessarily specify how that is implemented. It is best to decouple the model specification (the probabilistic program or simulator code) from the inference algorithm — much as we use a tool like **HistFactory** [88] to create a statistical model and then use **RooStats** [89] to provide generic statistical inference algorithms. Various inference engines have been developed implementing different inference strategies such as Importance Sampling [90] and specializations of Metropolis–Hastings [91] that are compatible with the complex latent space structure associated to stack traces. In general, the inference algorithms can be thought of as hijacking the random sampling inside of the simulator code to guide the simulator towards a certain output.

Early research in probabilistic programming required coding the simulator in special-purpose languages, which is not an attractive option for HEP as we have decades of work invested in our simulation code bases. Recently, however, the Etalumis project developed **PPX**, a cross-platform probabilistic execution protocol that allows an inference engine to control a simulator in a language-agnostic way [90, 92]. The Etalumis team integrated **PPX** into the **SHERPA** simulator and a simplified calorimeter simulation to demonstrate probabilistic programming with a real-world simulator (see Fig. 4).

The bulk of the probabilistic programming literature is phrased in terms of Bayesian statistics. The posterior distribution $p(z|x, \theta)$ is of no conceptual problem for an ardent, frequentist particle physicists, because while z may be latent, it is a random variable and

^eOften it is assumed that the quantity being conditioned on is directly sampled from a distribution with a known likelihood (conditioned on the latent state of the simulator at that point in the execution). Sometimes this is reasonable, but sometimes this assumption is violated and we want to condition on some more complicated function of the random variables with an intractable density. In that setting, one typically needs to introduce some tolerance or kernel. In this way, probabilistic programming can be seen as a more sophisticated and computationally efficient way of implementing approximate Bayesian computation.

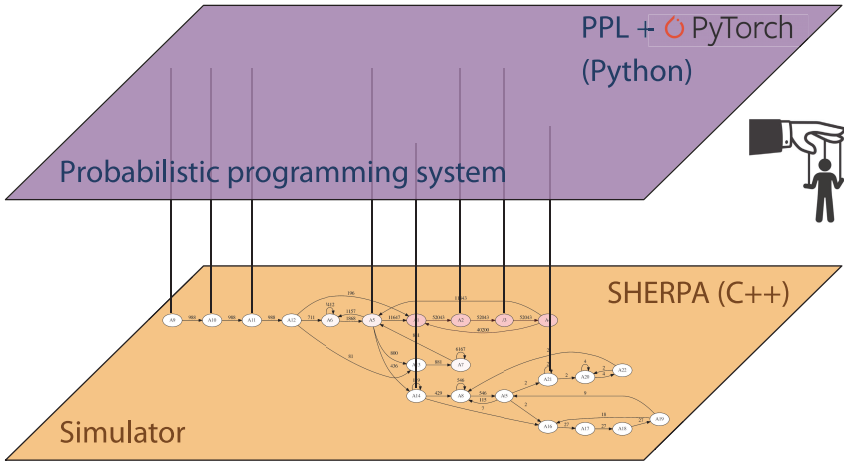


Fig. 4. An illustration of a Python-based probabilistic programming system’s inference engine controlling the **SHERPA** event generator through the **PPX** protocol. Figure taken from [92].

the joint distribution $p(x, z|\theta)$ is perfectly well defined. However, if one wanted to use probabilistic programming to infer the parameters of the simulation θ , then one would need to include a prior $p(\theta)$ and sample from that distribution at the beginning of the program. The result would be a probabilistic program for the joint model $p(x, z, \theta) = p(x, z|\theta)p(\theta)$, and one would then condition on x to obtain samples from the posterior $p(\theta, z|x)$ or the marginal $p(\theta|x)$.

6. Software and Computing

The methods described in this chapter are closely connected to the software and computing challenges of high-energy physics, particularly when we think about the high-luminosity LHC.

Initial results from phenomenological studies indicate that these new machine-learning based approaches provide substantial improvements in sensitivity to traditional approaches, but generating the training data is computationally expensive. However, with some additional work, the augmented data described in Sec. 2.4 can reduce the amount of simulated data needed by orders of magnitude. The Python library **MadMiner** [80] implements most of the machine

learning-based algorithms discussed in Secs. 2 and 3. It wraps around **MadGraph**, **Pythia**, and **Delphes** and thus automates the entire pipeline for a typical phenomenological study.^f The approach is compatible with a full simulation like **Geant4** as the necessary information can just be passed through the detector simulation similar to the weights used to assess uncertainty in the parton distribution functions. However, this still requires a modest investment in the experiments' simulation software.

The use of the learned likelihood ratio for reweighting event samples has the potential for a significant reduction in simulation costs as the reweighting factor can often be learned on parton-level or particle-level data without running the full simulation or reconstruction on large samples of simulated data with varied parameter settings. The CARL technique is being explored within ATLAS and integrated into the ATLAS software framework for this purpose [55].

Probabilistic programming also has the potential to address the computational resources needed for simulation at the high-luminosity LHC. Signs of new physics typically would hide in tails of background distributions, which are computationally expensive to populate with naive sampling approaches. HEP collaborations regularly use a form of importance sampling where the parton-level phase space is sliced (e.g. slices in the transverse momentum of outgoing partons to fill the high- p_T tail in the process $pp \rightarrow jj$). In this case, one merges several individual samples of simulated events weighted by N_s/σ_s , where N_s is the number of simulated samples and σ_s is total cross-section for that slice. However, this approach does not work for efficiently sampling regions of phase space that do not correspond to simple regions in the parton phase space. For example, consider the case where we want to populate the regions of phase space where standard QCD jets fake a boosted top. For a tagger based on a deep neural network [94] the fake rate is roughly 10^{-3} and much of the relevant fluctuations happen in the parton shower and are not reflected in the parton-level

^fThe **sbi** package [93] implements many simulation-based inference methods, in particular for Bayesian inference, in a problem-agnostic way, but does not provide any interface to particle physics simulators yet.

phase space. Event generators instrumented with probabilistic programming constructs offer the potential to efficiently sample these complicated regions of phase space, which is being explored with a simplified parton shower known as **Ginkgo** [95].

In the long term, we should not treat the simulation chain as a black box, but open them and begin to integrate automatic differentiation and probabilistic programming capabilities in them as that will enable more powerful and sample-efficient inference algorithms [14].

7. Summary

Particle physicists have a suite of simulators at their disposal that can model essentially all aspects of particle collisions with impressive fidelity. These tools use Monte Carlo methods to generate events, with the distribution of outputs depending on the parameters of the physics model. However, we cannot use these tools directly for inference because we cannot evaluate the probability of the simulator to generate a specific observed event. Because the likelihood is intractable, we cannot directly fit for the most likely parameter points or calculate exclusion limits from observed data. Historically, this challenge has been overcome by reducing the high-dimensional event data to one or two kinematic variables and using histograms or analytic functions to model the distribution of these observables. This makes inference possible, but often degrades the sensitivity of the analysis.

Here we reviewed simulation-based (or likelihood-free) inference methods that allow us to infer parameters based on high-dimensional event data. These methods are closely connected to other important tasks in HEP and provide the ability to reweight events [47, 54, 55], tune shower and detector-simulation parameters to data [54], unfold distributions [56], and detect anomalies [57], see Chapter 4. An important driver of these algorithms are the rapidly increasing capabilities of machine learning, which let us analyze high-dimensional data efficiently. In addition, extracting matrix-element information from the simulator and using it to augment training data can drastically reduce the number of simulations we need to run. We presented

algorithms based on these two ideas in which a neural network is trained as a surrogate for the likelihood or the likelihood ratio function or defines optimal observables, which can then be used in a traditional histogram-based analysis.

In first phenomenological LHC studies, these algorithms have been applied to Higgs precision measurements in vector boson fusion [7], in WH production [96], and in $t\bar{t}H$ production [80], as well as ZW measurements [63] and the search for massive resonances decaying into dijets [97]. The new machine learning-based techniques consistently led to more sensitive analyses than traditional histogram-based approaches such as simplified template cross-section measurements [96]. With a range of diagnostic tools and ideas for uncertainty quantification available and software packages making the application of these methods easier, the application of these new simulation-based inference techniques to data collected at the LHC experiments seems imminent.

Acknowledgments

We want to thank our collaborators Zubair Bhatti, Sally Dawson, Irina Espejo, Joeri Hermans, Samuel Homiller, Felix Kling, Gilles Louppe, Siddharth Mishra-Sharma, Juan Pavez, Sinclert Perez, Tilman Plehn, and Markus Stoye. This work was supported by the U.S. National Science Foundation (NSF) under the awards ACI-1450310, OAC-1836650, and OAC-1841471. We are grateful for the support of the Moore-Sloan data science environment at NYU.

References

- [1] F. James and M. Roos, Minuit: A system for function minimization and analysis of the parameter errors and correlations, *Comput. Phys. Commun.* **10** (1975) 343.
- [2] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H. S. Shao, T. Stelzer, P. Torrielli and M. Zaro, The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations, *J. High Energy Phys.* **2014** (2014) 79; arXiv:1405.0301 [hep-ph].

- [3] T. Gleisberg, S. Hoeche, F. Krauss, M. Schonherr, S. Schumann, F. Siegert and J. Winter, Event generation with SHERPA 1.1, *J. High Energy Phys.* **02** (2009) 7; arXiv:0811.4622 [hep-ph].
- [4] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen and P. Z. Skands, An introduction to PYTHIA 8.2, *Comput. Phys. Commun.* **191** (2015) 159; arXiv:1410.3012 [hep-ph].
- [5] G. Corcella, I. G. Knowles, G. Marchesini, S. Moretti, K. Odagiri, P. Richardson, M. H. Seymour and B. R. Webber, HERWIG 6: An event generator for hadron emission reactions with interfering gluons (including supersymmetric processes), *J. High Energy Phys.* **01** (2001) 10; arXiv:0011363 [hep-ph].
- [6] T. G. Collaboration, Geant4 — a simulation toolkit, *Nucl. Instrum. Methods Phys. Res. Sec. A* **506** (2003) 250.
- [7] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, A guide to constraining effective field theories with machine learning, *Phys. Rev. D* **98** (2018) 5, 052004; arXiv:1805.00020.
- [8] A. J. Barr and C. G. Lester, A review of the mass measurement techniques proposed for the large hadron collider, *J. Phys. G* **37** (2010) 123001; arXiv:1004.2732 [hep-ph].
- [9] P. Jackson, C. Rogan and M. Santoni, Sparticles in motion: Analyzing compressed SUSY scenarios with a new method of event reconstruction, *Phys. Rev. D* **95** (2017) 035031; arXiv:1607.08307 [hep-ph].
- [10] P. J. Diggle and R. J. Gratton, Monte Carlo methods of inference for implicit statistical models, *J. Roy. Statist. Soc. Ser. B (Methodological)* (1984).
- [11] K. Cranmer, Kernel estimation in high-energy physics, *Comput. Phys. Commun.* **136** (2001) 198; arXiv:hep-ex/0011057.
- [12] D. B. Rubin, Bayesianly justifiable and relevant frequency calculations for the applied statistician, *Ann. Statist.* **12** (1984) 1151.
- [13] M. A. Beaumont, W. Zhang and D. J. Balding, Approximate Bayesian computation in population genetics, *Genetics* **162** (2002) 2025.
- [14] K. Cranmer, J. Brehmer and G. Louppe, The frontier of simulation-based inference, *Proc. Natl. Acad. Sci. USA* **117**(48) (2020) 30055–30062; arXiv:1911.01429 [stat.ML].
- [15] J. Brehmer, K. Cranmer, I. Espejo, F. Kling, G. Louppe and J. Pavez, Effective LHC measurements with matrix elements and machine learning, in *19th Int. Workshop on Advanced Computing and Analysis Techniques in Physics Research: Empowering the revolution: Bringing Machine Learning to High Performance Computing (ACAT 2019)*, Saas-Fee, Switzerland, March 11–15, 2019 (2019); arXiv:1906.01578.
- [16] K. Cranmer, Practical statistics for the LHC, in *Proc. 2011 European School of High-Energy Physics (ESHEP 2011)*, Cheile Gradistei, Romania, September 7–20, 2011 (2015); arXiv:1503.07622 [physics.data-an].
- [17] K. Kondo, Dynamical likelihood method for reconstruction of events with missing momentum: I. Method and toy models, *J. Phys. Soc. Japan* **57** (1988) 4126.

- [18] V. M. Abazov and Others, A precision measurement of the mass of the top quark, *Nature* **429** (2004) 638; arXiv:hep-ex/0406031.
- [19] P. Artoisenet and O. Mattelaer, MadWeight: Automatic event reweighting with matrix elements, *Proc. Sci. CHARGED200* (2008) 25.
- [20] Y. Gao, A. V. Gritsan, Z. Guo, K. Melnikov, M. Schulze and N. V. Tran, Spin determination of single-produced resonances at hadron colliders, *Phys. Rev. D* **81** (2010) 75022; arXiv:1001.3396 [hep-ph].
- [21] J. Alwall, A. Freitas and O. Mattelaer, Matrix element method and QCD radiation, *Phys. Rev. D* **83** (2011) 74010; arXiv:1010.2263 [hep-ph].
- [22] S. Bolognesi, Y. Gao, A. V. Gritsan, K. Melnikov, M. Schulze, N. V. Tran and A. Whitbeck, On the spin and parity of a single-produced resonance at the LHC, *Phys. Rev. D* **86** (2012) 95031; arXiv:1208.4018 [hep-ph].
- [23] P. Avery, D. Bourilkov, M. Chen, T. Cheng, A. Drozdetskiy, J. S. Gainer, A. Korytov, K. T. Matchev, P. Milenovic, G. Mitselmakher, M. Park, A. Rinkevicius and M. Snowball, Precision studies of the Higgs boson decay channel $H \rightarrow ZZ \rightarrow 4l$ with MEKD, *Phys. Rev. D* **87** (2013) 55006; arXiv:1210.0896 [hep-ph].
- [24] J. R. Andersen, C. Englert and M. Spannowsky, Extracting precise Higgs couplings by using the matrix element method, *Phys. Rev. D* **87** (2013) 15019; arXiv:1211.3011 [hep-ph].
- [25] J. M. Campbell, R. K. Ellis, W. T. Giele and C. Williams, Finding the Higgs boson in decays to $Z\gamma$ using the matrix element method at next-to-leading order, *Phys. Rev. D* **87** (2013) 73005; arXiv:1301.7086 [hep-ph].
- [26] P. Artoisenet, P. de Aquino, F. Maltoni and O. Mattelaer, Unravelling $t\bar{t}h$ via the matrix element method, *Phys. Rev. Lett.* **111** (2013) 91802; arXiv:1304.6414 [hep-ph].
- [27] J. S. Gainer, J. Lykken, K. T. Matchev, S. Mrenna and M. Park, The matrix element method: Past, present, and future, in *Proceedings, 2013 Community Summer Study on the Future of U.S. Particle Physics: Snowmass on the Mississippi (CSS2013)*, Minneapolis, MN, USA, July 29–August 6, 2013 (2013); arXiv:1307.3546.
- [28] D. Schouten, A. Deabreu and B. Stelzer, Accelerated matrix element method with parallel computing, *Comput. Phys. Commun.* **192** (2015) 54; arXiv:1407.7595 [physics.comp-ph].
- [29] T. Martini and P. Uwer, Extending the matrix element method beyond the born approximation: Calculating event weights at next-to-leading order accuracy, *J. High Energy Phys.* **2015** (2015) 83; arXiv:1506.08798 [hep-ph].
- [30] A. V. Gritsan, R. Röntsch, M. Schulze and M. Xiao, Constraining anomalous Higgs boson couplings to the heavy-flavor fermions using matrix element techniques, *Phys. Rev. D* **94** (2016) 55023; arXiv:1606.03107 [hep-ph].
- [31] T. Martini and P. Uwer, The matrix element method at next-to-leading order QCD for hadronic collisions: Single top-quark production at the LHC as an example application (2017); arXiv:1712.04527 [hep-ph].
- [32] M. Kraus, T. Martini and P. Uwer, Predicting event weights at next-to-leading order QCD for jet events defined by $2 \rightarrow 1$ jet algorithms (2019); arXiv:1901.08008 [hep-ph].

- [33] D. E. Soper and M. Spannowsky, Finding physics signals with shower deconstruction, *Phys. Rev. D* **84** (2011) 74002; arXiv:1102.3480 [hep-ph].
- [34] D. E. Soper and M. Spannowsky, Finding top quarks with shower deconstruction, *Phys. Rev. D* **87** (2013) 54012; arXiv:1211.3140 [hep-ph].
- [35] D. E. Soper and M. Spannowsky, Finding physics signals with event deconstruction, *Phys. Rev. D* **89** (2014) 94005; arXiv:1402.1189 [hep-ph].
- [36] C. Englert, O. Mattelaer and M. Spannowsky, Measuring the Higgs-bottom coupling in weak boson fusion, *Phys. Lett. B* **756** (2016) 103; arXiv:1512.03429 [hep-ph].
- [37] L. Holmstrom, S. Sain and H. Miettinen, A New multivariate technique for top quark search, *Comput. Phys. Commun.* **88** (1995) 195.
- [38] D. J. Rezende and S. Mohamed, Variational inference with normalizing flows, in *32nd Int. Conf. Machine Learning, ICML 2015* (2015) 1530; arXiv:1505.05770.
- [39] G. Papamakarios, E. Nalisnick, D. Jimenez Rezende, S. Mohamed and B. Lakshminarayanan, Normalizing flows for probabilistic modeling and inference (2019); arXiv:1912.02762 [stat.ML].
- [40] K. Cranmer and G. Louppe, Unifying generative models and exact likelihood-free inference with conditional bijections, *J. Brief Ideas* (2016); doi:10.5281/zenodo.198541.
- [41] K. Cranmer, J. Pavez and G. Louppe, Approximating likelihood ratios with calibrated discriminative classifiers (2015); arXiv:1506.02169.
- [42] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski and D. Whiteson, Parameterized neural networks for high-energy physics, *Eur. Phys. J. C* **76** (2016) 235; arXiv:1601.07913 [hep-ex].
- [43] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, A. Rousselot, R. Winterhalder, L. Ardizzone and U. Köthe, Invertible networks or partons to detector and back again (2020); arXiv:2006.06685 [hep-ph].
- [44] B. Nachman and D. Shih, Anomaly detection with density estimation, *Phys. Rev. D* **101** (2020) 75042; arXiv:2001.04990 [hep-ph].
- [45] R. M. Neal, Computing likelihood functions for high-energy physics experiments when distributions are defined by simulators with nuisance parameters, in *Proc. PHYSTAT LHC Workshop on Statistical Issues for LHC Physics, PHYSTAT 2007* (2008); <http://cds.cern.ch/record/1099977/files/p111.pdf>.
- [46] Y. Fan, D. J. Nott and S. A. Sisson, Approximate Bayesian computation via regression density estimation, *Stat.* **2** (2013) 34; arXiv:1212.1479 [stat.CO].
- [47] K. Cranmer, NIPS 2016 keynote: Machine learning & likelihood free inference in particle physics (2016); doi:10.6084/m9.figshare.4291565.v1.
- [48] G. Louppe, K. Cranmer and J. Pavez, Carl: A likelihood-free inference toolbox, *J. Open Source Softw.* **1** (2016) 11.
- [49] S. Mohamed and B. Lakshminarayanan, Learning in implicit generative models (2016); arXiv:1610.03483.
- [50] O. Thomas, R. Dutta, J. Corander, S. Kaski and M. U. Gutmann, Likelihood-free inference by ratio estimation (2016); arXiv:1611.10242.

- [51] M. U. Gutmann, R. Dutta, S. Kaski and J. Corander, Likelihood-free inference via classification, *Statist. Comput.* **28** (2018) 411; arXiv:1407.4981.
- [52] T. Dinev and M. U. Gutmann, Dynamic likelihood-free inference via ratio estimation (DIRE) (2018); arXiv:1810.09899.
- [53] J. Hermans, V. Begy and G. Louppe, Likelihood-free MCMC with approximate likelihood ratios (2019); arXiv:1903.04057 [stat.ML].
- [54] A. Andreassen and B. Nachman, Neural networks for full phase-space reweighting and parameter tuning, *Phys. Rev. D* **101** (2020) 91901; arXiv:1907.08209 [hep-ph].
- [55] L. Vesterbacka, carl-torch, (2020); doi:10.5281/zenodo.4062049.
- [56] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman and J. Thaler, OmniFold: A method to simultaneously unfold all observables, *Phys. Rev. Lett.* **124** (2020) 182001; arXiv:1911.09107 [hep-ph].
- [57] A. Andreassen, B. Nachman and D. Shih, Simulation assisted likelihood-free anomaly detection, *Phys. Rev. D* **101** (2020) 95004; arXiv:2001.05001 [hep-ph].
- [58] J. Brehmer, G. Louppe, J. Pavez and K. Cranmer, Mining gold from implicit models to improve likelihood-free inference, *Proc. Natl. Acad. Sci. USA* **117** (2020) 5242; arXiv:1805.12244.
- [59] J. Brehmer, S. Mishra-Sharma, J. Hermans, G. Louppe and K. Cranmer, Mining for dark matter substructure: Inferring subhalo population properties from strong lenses with machine learning, *Astrophys. J.* **886** (2019) 49; arXiv:1909.02005.
- [60] J. Brehmer, K. Cranmer, G. Louppe and J. Pavez, Constraining effective field theories with machine learning, *Phys. Rev. Lett.* **121** (2018) 111801; arXiv:1805.00013.
- [61] M. Stoye, J. Brehmer, G. Louppe, J. Pavez and K. Cranmer, Likelihood-free inference with an improved cross-entropy estimator, in *NeurIPS Workshop on Machine Learning for the Physical Sciences* (2019); arXiv:1808.00973.
- [62] G. Aad *et al.*, A morphing technique for signal modelling in a multidimensional space of coupling parameters (2015); <https://cds.cern.ch/record/2066980>.
- [63] S. Chen, A. Glioti, G. Panico and A. Wulzer, Parametrized classifiers for optimal EFT sensitivity (2020); arXiv:2007.10356 [hep-ph].
- [64] G. Papamakarios, D. C. Sterratt and I. Murray, Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows, in *Int. Conf. Artificial Intelligence and Statistics* (2018); arXiv:1805.07226.
- [65] J.-M. Lueckmann, G. Bassetto, T. Karaletsos and J. H. Macke, Likelihood-free inference with emulator networks, in *Proc. 1st Symposium on Advances in Approximate Bayesian Inference*, eds. F. Ruiz, C. Zhang, D. Liang, and T. Bui, (PMLR, 2018); arXiv:1805.09294.
- [66] C. Durkan, I. Murray and G. Papamakarios, On contrastive learning for likelihood-free inference (2020); arXiv:2002.03712 [stat.ML].
- [67] P. Ranjan, D. Bingham and G. Michailidis, Sequential experiment design for contour estimation from complex computer codes, *Technometrics* **50** (2008) 527.

- [68] J. Bect, D. Ginsbourger, L. Li, V. Picheny and E. Vazquez, Sequential design of computer experiments for the estimation of a probability of failure, *Statist. Comput.* **22** (2012) 773.
- [69] L. Heinrich, G. Louppe and K. Cranmer, diana-hep/excursion: Initial Zenodo Release, (2018); doi:10.5281/zenodo.1634428.
- [70] P. de Castro and T. Dorigo, INFERNO: Inference-aware neural optimisation, *Comput. Phys. Commun.* **244** (2019) 170; arXiv:1806.04743.
- [71] L. Heinrich and N. Simpson, pyhf/neos, (2020); doi:10.5281/zenodo.3697981.
- [72] G. Cowan, K. Cranmer, E. Gross and O. Vitells, Asymptotic formulae for likelihood-based tests of new physics, *Eur. Phys. J. C* **71** (2011) 1554; arXiv:1007.1727.
- [73] J. Alsing and B. Wandelt, Generalized massive optimal data compression, *Mon. Notices Roy. Astron. Soc. Lett.* **476** (2018) L60; arXiv:1712.00012.
- [74] J. Alsing, B. Wandelt and S. Feeney, Massive optimal data compression and density estimation for scalable, likelihood-free inference in cosmology, *Mon. Notices Roy. Astron. Soc.* **477** (2018) 2874; arXiv:1801.01497.
- [75] D. Atwood and A. Soni, Analysis for magnetic moment and electric dipole moment form factors of the top quark via $e^+e^- \rightarrow t\bar{t}$, *Phys. Rev. D* **45** (1992) 2405.
- [76] M. Davier, L. Duflot, F. Le Diberder and A. Rougé, The optimal method for the measurement of tau polarization, *Phys. Lett. B* **306** (1993) 411.
- [77] M. Diehl and O. Nachtmann, Optimal observables for the measurement of three gauge boson couplings in $e^+e^- \rightarrow W^+W^-$, *Zeit. Phys. C Particles Fields* **62** (1994) 397.
- [78] J. Brehmer, K. Cranmer, F. Kling and T. Plehn, Better Higgs boson measurements through information geometry, *Phys. Rev. D* **95** (2017) 73002; arXiv:1612.05261 [hep-ph].
- [79] J. Brehmer, F. Kling, T. Plehn and T. M. Tait, Better Higgs-CP tests through information geometry, *Phys. Rev. D* **97** (2018) 9; arXiv:1712.02350 [hep-ph].
- [80] J. Brehmer, F. Kling, I. Espejo and K. Cranmer, MadMiner: Machine learning-based inference for particle physics, *Comput. Softw. Big Sci.* **4** (2020) 1; arXiv:1907.10621.
- [81] B. Nachman and C. Shminin, AI safety for high energy physics (2019); arXiv:1910.08606 [hep-ph].
- [82] G. Louppe, M. Kagan and K. Cranmer, Learning to pivot with adversarial networks, *Adv. Neural Inform. Process. Syst.* (2017) 981.
- [83] J. Alsing and B. Wandelt, Nuisance hardened data compression for fast likelihood-free inference, *Mon. Notices Roy. Astron. Soc.* **488** (2019) 5093; arXiv:1903.01473.
- [84] S. S. Wilks, The large-sample distribution of the likelihood ratio for testing composite hypotheses, *Ann. Math. Statist.* **9** (1938) 60.
- [85] A. Wald, Tests of statistical hypotheses concerning several parameters when the number of observations is large, *Trans. Amer. Math. Soc.* **54** (1943) 426.
- [86] A. D. Gordon, T. A. Henzinger, A. V. Nori and S. K. Rajamani, Probabilistic programming, in *Proc. Future of Software Engineering, FOSE 2014* (2014).

- [87] F. Wood, J. W. Van De Meent and V. Mansinghka, A new approach to probabilistic programming inference, *J. Machine Learn. Res.* (2014); arXiv:1507.00996.
- [88] K. Cranmer, G. Lewis, L. Moneta, A. Shibata and W. Verkerke, HistFactory: A tool for creating statistical models for use with RooFit and RooStats (2012); <https://cds.cern.ch/record/1456844>.
- [89] L. Moneta, K. Belasco, K. S. Cranmer, S. Kreiss, A. Lazzaro, D. Piparo, G. Schott, W. Verkerke and M. Wolf, The RooStats project, *Proc. Sci. ACAT2010* (2010) 057; arXiv:1009.1003 [physics.data-an].
- [90] A. G. Baydin, L. Shao, W. Bhimji, L. Heinrich, L. Meadows, J. Liu, A. Munk, S. Naderiparizi, B. Gram-Hansen, G. Louppe, M. Ma, X. Zhao, P. Torr, V. Lee, K. Cranmer, Prabhat and F. Wood, Etalumis: Bringing probabilistic programming to scientific simulators at scale, in *Int. Conf. High Performance Computing, Networking, Storage and Analysis, SC* (2019); arXiv:1907.03382.
- [91] D. Wingate, A. Stuhlmueeller and N. Goodman, Lightweight implementations of probabilistic programming languages via transformational compilation, in *Proc. JMLR Workshop and Conf.* 11–13 April (2011).
- [92] A. G. Baydin *et al.*, Efficient probabilistic inference in the quest for physics beyond the standard model, in *Advances in Neural Information Processing Systems* (2019).
- [93] A. Tejero-Cantero, J. Boelts, M. Deistler, J.-M. Lueckmann, C. Durkan, P. J. Gonçalves, D. S. Greenberg and J. H. Macke, sbi: A toolkit for simulation-based inference, *J. Open Source Softw.* **5** (2020) 2505.
- [94] A. Butter *et al.*, The machine learning landscape of top taggers, *SciPost Phys.* **7** (2019) 014; arXiv:1902.09914 [hep-ph].
- [95] K. Cranmer, S. Macaluso and D. Pappadopulo, Ginkgo: A simplified parton shower (2019); <https://github.com/sebastianmacaluso/toyjetsshower>.
- [96] J. Brehmer, S. Dawson, S. Homiller, F. Kling and T. Plehn, Benchmarking simplified template cross sections in WH production, *J. High Energy Phys.* **2019** (2019) 11; arXiv:1908.06980 [hep-ph].
- [97] J. Hollingsworth and D. Whiteson, Resonance searches with machine learned likelihood ratios (2020); arXiv:2002.04699 [hep-ph].

This page intentionally left blank

Chapter 17

Dealing with Nuisance Parameters

T. Dorigo* and P. de Castro Manzano†

*Istituto Nazionale di Fisica Nucleare–Sezione di Padova,
Via Marzolo 8, 35131 Padova, Italy*

**tommaso.dorigo@cern.ch*

†pablo.de.castro@cern.ch

In this chapter we consider the impact of nuisance parameters on the effectiveness of machine learning in high-energy physics problems, and discuss techniques that allow to include their effect and reduce their impact in the search for optimal selection criteria and variable transformations. The introduction of nuisance parameters complicates the supervised learning task and its correspondence with the data analysis goal, due to their contribution degrading model performances in real data, and the necessary addition of uncertainties in the resulting statistical inference. The approaches discussed include nuisance-parameterized models, modified or adversary losses, semi-supervised learning approaches, and inference-aware techniques.

1. Introduction

As was demonstrated in previous chapters, particle physics offers a variety of use cases for machine learning techniques. Of these, probably the most common is the use of supervised classification to construct low-dimensional event summaries, which allow to perform statistical inference on the parameters of interest. The learned *summary statistics* — functions of the data that are informative about the relevant properties of the data — can efficiently combine high-dimensional information from each event into one or a few variables which may be used as the basis of statistical inference. The informational source of this compression are simulated observations produced

by a complex generative model; the latter reproduces the chain of physical processes occurring in subnuclear collisions and the subsequent interaction of the produced final state particles with detection elements.

The fidelity of the event description provided by the simulation is usually limited, due to a variety of factors. These may include imperfections in the modeling of the physical processes employed by the simulation (such as the use of a leading order approximation for the hard scattering process), limited precision in the description of detector response (usually due to imperfect knowledge of the relevant calibration constants), uncertainty in fundamental physics parameters liable to condition the observations (e.g., the mass of a decaying particle), or simply a consequence of the finiteness of the number of simulated observations in a given region of feature space. To account for these “known unknowns”, commonly referred to as *nuisance parameters* in statistics literature, the model needs to be enlarged by the inclusion of corresponding variables which are not of direct relevance, yet have to be considered during inference in order to make calibrated statements about the parameters of interest.

Because simulated observations are also at the basis of the construction of the likelihood function or any other estimator employed for the extraction of the wanted information, either directly or through intermediate summary statistics constructed with them, nuisance parameters must be directly included in the statistical model. The inclusion of the effect of nuisances results in the enlargement of confidence intervals on the parameters of interest; nuisance parameters are correspondingly referred to as systematic uncertainties. The effect is not exclusive to parameter estimation problems: it also commonly arises in hypothesis testing problems, such as when a test of the presence of a new physics signal is performed on data otherwise conforming to the Standard Model hypothesis. The presence of nuisance parameters then causes a reduction of the statistical power of the test. Nuisance parameters are therefore one of main factors limiting the precision and discovery reach of HEP analyses. However, it is

worth stressing that they are not intrinsically problematic: they may be viewed as a useful tool that allows us to model those uncertainties associated to not having an exact model for the data.

The mentioned hindrances apply in the same manner to analyses which employ machine learning algorithms to reduce the dimensionality of the data: the effect of nuisance parameters must be accounted for in statistical inference based on summary statistics constructed from the output of the summarizing function. Neither the training loss nor the standard measures of performance for the learning task itself are aligned with the inference goal when the simulated observations depend on additional modeling parameters that are unknown. This chapter focuses on issues arising from the application of machine learning techniques to problems where nuisance parameters are relevant, and on different approaches that have been proposed to overcome the resulting limitations.

1.1. *Probabilistic classification as density ratio estimation*

Before delving into the subject matter, it is important to review the relation between the learning tasks performed in HEP data analysis and the statistical properties of training data. As introduced earlier, machine-learning based data transformations in HEP are often based on probabilistic classification models trained with samples from computer simulations of the different processes. The simplest way to understand a probabilistic classifier is in terms of probability density ratios.

By training a probabilistic classification model (e.g., a neural network optimizing binary cross entropy, BCE) to distinguish samples of simulated signal (labeled by $y = 1$) and background (labeled by $y = 0$), we are approximating the density ratio $r(\mathbf{x}) = p(\mathbf{x}|y = 1)/p(\mathbf{x}|y = 0)$ between the signal and background generating distributions $p(\mathbf{x}|y = 1)$, $p(\mathbf{x}|y = 0)$. For example, when using BCE as a loss function, the density ratio $r(\mathbf{x})$ can be approximated using the

classification output $c(\mathbf{x})$ by virtue of the following relation:

$$\frac{c(\mathbf{x})}{1 - c(\mathbf{x})} \approx \frac{p(y = 1|\mathbf{x})}{p(y = 0|\mathbf{x})} = \frac{p(\mathbf{x}|y = 1)}{p(\mathbf{x}|y = 0)} \frac{p(y = 1)}{p(y = 0)} = r(\mathbf{x}) \frac{p(y = 1)}{p(y = 0)}, \quad (1)$$

where $p(y = 1)/p(y = 0)$ is independent of \mathbf{x} , and may be simply estimated as the ratio between the total number of observations from each category in the training dataset. The previous approximation becomes an exact equality only for the best possible classifier, the so-called *Bayes optimal classifier*, which is a function of the true and generally unknown density ratio between the generating distributions of signal and background in training data. In practice, good approximations can be obtained given enough data, flexible models, and a powerful learning rule. The previous relation is not unique for BCE-based probabilistic classification models, as it also holds for other approaches that minimize continuous relaxations of the zero-one loss and could be generalized for the multi-class case.

Viewing probabilistic classifiers as probabilistic density ratio estimators allows us to abstract away from the specific machine learning techniques used to construct the model (e.g., gradient boosting or neural networks trained by stochastic gradient descent), and also provides a clear theoretical definition for the best possible classifier: that is the one that minimises the risk or generalization error of a classification problem as a function of the probability density ratios between the data-generating distributions. Furthermore, density ratios can also be easily linked with the statistical inference goals of data analysis and may effectively be used to study the limitations of machine learning approaches.

We may explore the previous formulation in the case when the generating distributions of data are not fully specified, but depend on additional unknown nuisance parameters $\boldsymbol{\theta}$. A classifier distinguishing samples from the data-generating distributions $p(\mathbf{x}|\boldsymbol{\theta}, y = 1)$ and $p(\mathbf{x}|\boldsymbol{\theta}, y = 0)$ will still be approximating a function of the density ratio

$$r(\mathbf{x}; \boldsymbol{\theta}) = \frac{p(\mathbf{x}|\boldsymbol{\theta}, y = 1)}{p(\mathbf{x}|\boldsymbol{\theta}, y = 0)} \quad (2)$$

and hence will itself depend on the actual value of the parameters θ . If we assume that the true value of the parameters is fixed but unknown (which is the typical setting used for frequentist inference in HEP), then the optimal classifier is not uniquely defined.^a For example, a classifier trained using simulated data generated for specific parameters θ_s might not be optimal at classifying experimental data observations that correspond to the unknown parameter value $\theta_d \neq \theta_s$. This is the main issue with nuisance parameters from the perspective of the machine learning performance.

1.2. *Nuisance parameters in statistical inference*

Another challenge with nuisance parameters, arguably the most relevant one, is the way they affect our ability to extract useful information about our models of nature from experimental data when carrying out statistical inference in the form of hypothesis testing or interval estimation. Nuisance parameters must be viewed as necessary instruments to make unbiased and calibrated statistical inference statements when we do not have a perfectly known model of the data. Their effect is not solely a complication for analyses based on machine learning approaches, as it also applies to analysis based on manual variable transformations; however, as will be discussed in detail in Sec. 5, the presence of nuisance parameters can put into question the role of supervised learning models in the context of statistical inference, voiding them of the standard advantages that otherwise make them so apt for dimensional reduction in physical analyses.

The previous concerns are closely related to the misalignment between the goal of particle physics analyses — to infer information about our models of nature given the data — and the classification and regression objectives of supervised learning approaches.

^aIn a Bayesian setting, if the parameters are treated as random variables associated with a prior probability density distribution $\pi(\theta)$, then the optimal Bayes classifier can be uniquely defined, because parameter sampling from $\pi(\theta)$ may be considered part of the data generating procedure so the density ratio can be implicitly marginalized.

As reviewed in Sec. 5.1, probabilistic classification models offer principled guarantees of optimality of the inference goal in inference problems based on mixture models where the mixture fraction is the only parameter of interest in the absence of nuisance parameters. In general terms, the supervised learning task can be considered a proxy objective to obtain low-dimensional data transformations that are still informative about the parameters of interest.

In this context, it is worth introducing the concept of *summary statistic*. For a set of n independent and identically distributed (i.i.d.) observations or events $D = \{\mathbf{x}_0, \dots, \mathbf{x}_n\}$, where each $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ is a d -dimensional representation of the event information at an arbitrary representational level (e.g., raw detector readout, physical objects, or a subset of columnar variables), a summary statistic of the data D is simply a function of the data that reduces their dimensionality. An infinite number of different summary statistics can be constructed for a given set of data, but we are generally only interested in those which are as low-dimensional as possible while preserving as much as possible the information relevant for the statistical inference goal of a given analysis. The low-dimensionality requirement is made necessary by the curse of dimensionality, due to $p(\mathbf{x}|\boldsymbol{\theta})$ not being known analytically and having to be estimated from a finite number of simulated observations.

Most of the operations that reduce the dimensionality of experimental data, either in terms of reducing the number of events (e.g., trigger and event selection) or its representation (reconstruction, physical object selection, feature engineering, multivariate methods, histograms) can be viewed through the lens of summary statistics (see [1, Chapter 3]). Summary statistics used in high-energy physics are thus often a composition of several transformations, yet for the purpose of machine learning models we are usually interested the final components of the type $\mathbf{s}(D) = \{\mathbf{s}(\mathbf{x}_i) \mid \forall \mathbf{x}_i \in D\}$ that are the product of the event-wise application of a function

$$\mathbf{s}(\mathbf{x}) : \mathcal{X} \subseteq \mathbb{R}^d \longrightarrow \mathcal{Y} \subseteq \mathbb{R}^b \quad (3)$$

reducing the dimensionality of each event from the original feature space $\mathcal{X} \subseteq \mathbb{R}^d$, which could be already a transformation of the detector readout or set of engineered features, to a new low-dimensional space $\mathcal{Y} \subseteq \mathbb{R}^b$. Such a transformation may be used to reduce the data dimensionality from $n \times d$ to $n \times b$. Since the observations are assumed i.i.d, if b is very small we use simulated observations to estimate the probability density $p(\mathbf{s}(\mathbf{x})|\boldsymbol{\theta})$ by non-parametric means to carry out the inference goal. Most commonly in HEP applications, even simpler sample-wise statistics are instead constructed from $\mathbf{s}(D)$, such as the number of observations for which $\mathbf{s}(\mathbf{x})$ is within a given range (in so-called cut-and-count analyses) or simply a histogram of $\mathbf{s}(\mathbf{x})$. For these simpler statistics, sets of simulations of each process produced with different values of the nuisance parameters are interpolated to model the effect of the nuisances, enabling the construction of likelihoods based on the product of Poisson terms.

The choice of the dimensionality reducing transformation determines the inference reach of a given analysis. Choosing a summary statistic is not easy even in the absence of nuisance parameters, since naive choices of data transformation can very easily lead to a significant loss of useful information about the parameter of interest. Machine learning models, in particular probabilistic classification models trained to distinguish observations from different processes, are increasingly being used as an automated way to obtain summary statistic transformations. This is because the output of probabilistic classifiers approximates density ratios as discussed in Sec. 1.1. For simple hypothesis testing, density ratios are closely related to the optimal likelihood ratio test statistic, in the case of parameter estimation for the problem of inference of mixture fractions in the absence of nuisance parameters, as discussed in detail in Sec. 5.1.

It is of benefit for the ensuing discussion to succinctly recall how nuisance parameters can be “profiled away” from a likelihood function in the extraction of confidence intervals on parameters of interest; for a more general discussion of how the effect of nuisance parameters is accounted for in physics measurements, see e.g. [2–4].

We consider the measurement of a physical quantity in statistical terms as a problem of parameter estimation, whose solution relies on the specification of a statistical model wherein those quantities appear as free parameters. Under the assumption that experimental data conform to the specified model, the measurement may be carried out by constructing suitable estimators for the parameters of interest, which here we formulate through the specification of a likelihood function. Letting θ identify the parameters of interest, α describe systematic uncertainties affecting the model, and x_i , $i = 1, \dots, N$, be the collected data, understood to be a set of N random i.i.d. variables, the joint probability density can be written as $p(x, \theta, \alpha)$. This enables the construction of a likelihood function

$$\mathcal{L}(\theta, \alpha) = \prod_{i=1}^N p(x_i, \theta, \alpha). \quad (4)$$

If nuisance parameters α were absent in the above formulation, one would proceed directly to construct estimators of the parameters of interest as

$$\hat{\theta} = \arg \max_{\theta} \mathcal{L}(\theta). \quad (5)$$

The dependence on α can be dealt with by first obtaining the profile of the likelihood function, $\text{PL}(\theta)$, by maximizing \mathcal{L} as a function of the nuisances,

$$\text{PL}(\theta) = \sup_{\alpha} \mathcal{L}(\theta, \alpha), \quad (6)$$

and then proceeding as above. Uncertainties in the parameters of interest may be extracted from the curvature of the profile likelihood at its maximum exactly as is done with \mathcal{L} in the general case [5, 6]. This “profile likelihood method” is conceptually simple and practical to implement^b if the likelihood is differentiable with respect to the parameters, but may meet with technical problems (e.g., a slow

^bIn particle physics practice a widely used implementation is the MINUIT package [7], which offers profile likelihood evaluation through the MIGRAD routine.

convergence) as well as intractability for high dimensional α . The same issues affect in general the main alternative solution, which consists in computing the marginalized likelihood $\mathcal{L}_m(\theta)$ as

$$\mathcal{L}_m(\theta) = \int \mathcal{L}(\theta, \alpha) p(\alpha) d\alpha. \quad (7)$$

In both cases, knowledge of the PDF of nuisance parameters $p(\alpha)$ is mandatory for a meaningful solution. In a Bayesian construction p may be a subjective prior for the nuisance parameters; in general it may be the result of an external constraint, such as an independent measurement. Whatever the source, any imprecision in $p(\alpha)$ will in general affect the parameter estimates with increases in bias and/or variance.

1.3. Toward fully sufficient statistic summaries

The reduction of *statistical* uncertainty on the estimate of parameters of interest is a common goal of machine learning applications in experimental HEP. A suitable summary statistic may be obtained by training a high-performance boosted decision tree or an artificial neural network on simulated sets of data. The summary enables the extraction of the highest possible amount of information on the unknown true values of the physical quantities under measurement, *conditional on the validity of the underlying physics model* used to generate the training samples, as well as of specific assumptions on the value of relevant nuisance parameters. The crucial conditional clause above is usually hard to get rid of, because of the complexity of the problems, their high dimensionality, the typically unknown PDF of nuisance parameters, and/or the effects those parameters have on the physical model. When any one of the above effects play a role, the obtained summary statistic cannot be *sufficient*: being oblivious to a part of the feature space, it does not retain all the information contained in the data relevant to the parameter estimation task, and is therefore liable to be outperformed.

Notwithstanding the ubiquity of the above stated problem, the adjective *optimal* is often employed when reporting physics analysis results, usually in connection with incremental advances over

state-of-the-art of the employed techniques, for the common use case of classification performance in signal vs. background discrimination problems. The classical justification for a claim that a chosen algorithm or architecture and its output (a classification score) be *optimal* for the measurement task at hand is based on examination of associated performance measures such as the integral of the Receiver Output Characteristic curve (see Chapter 2), or on background acceptance estimates at fixed purity — or *vice versa*. In the absence of nuisance parameters those figures of merit are generally effective as a proxy to classification performance, when their maximization closely tracks the theoretical minimum value of the statistical uncertainty on the intermediate physical parameter to which the classification algorithm is sensitive, such as, e.g., a signal fraction. Yet they are blind to the more general problem connected with the subsequent extraction of, say, the cross-section of a physical reaction contributing to events labeled as signal, when uncertainties of non-stochastic nature are included.

A simple toy example may help pointing out the typical issues. Let us suppose that a dataset includes events originated from a signal process of interest S , in addition to ones coming from a known background source B . The typical output of a classifier trained to distinguish the event classes may be the one shown in Fig. 1 (left), which we parameterized using exponential functions with normalized density functions in the $x \in [0, 1]$ range:

$$S(x) = \frac{e^x}{e - 1}, \quad (8)$$

$$B(x) = \frac{\alpha e^{-\alpha x}}{1 - e^{-\alpha}}. \quad (9)$$

Above, we have included a nuisance parameter α in the definition of the background PDF. While the variations described by α in this example are very simple, the typical situation they mimic is a very common one, as the background distribution in HEP analysis problems is usually affected by significant uncertainties on its shape; α therefore describes what would be called a “background-shape

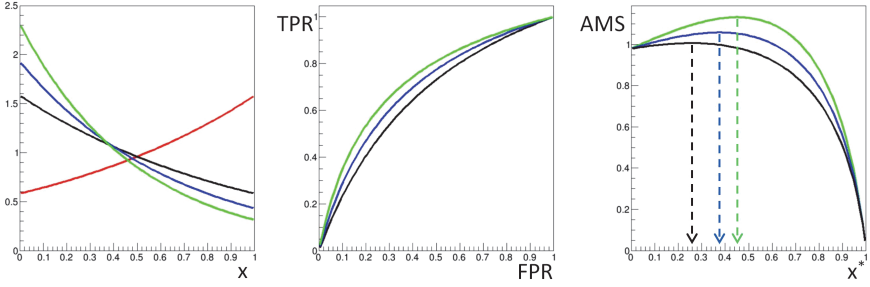


Fig. 1. A simple toy classification model. Left: the PDF of a signal process (red) is compared to the PDF of background for three values of the nuisance parameter, $\alpha = 1.0, 1.5, 2.0$; center: ROC curves corresponding to the three background distributions; right: values of the AMS figure of merit (see text for details) as a function of the selection cut value x^* . The location of maxima are shown by the corresponding arrows.

systematic uncertainty”. If we define the true positive rate (TPR) and false positive rate (FPR) of a selection criterion $x > x^*$ as

$$\text{TPR}(x^*) = \int_{x^*}^1 S(x)dx = \frac{e - e^{x^*}}{e - 1}, \quad (10)$$

$$\text{FPR}(x^*) = \int_{x^*}^1 B(x)dx = \frac{e^{-\alpha x^*} - e^{-\alpha}}{1 - e^{-\alpha}}, \quad (11)$$

then with simple algebra we may derive the ROC curve as the functional dependence of TPR on FPR:

$$\text{TPR}(\text{FPR}) = \frac{e - [e^{-\alpha} + (1 + e^{-\alpha})\text{FPR}]^{-1/\alpha}}{e - 1} \quad (12)$$

By examining the shape of ROC curves resulting from different values of the nuisance parameter α (see Fig. 1, center), one may verify the qualitative benefit of $B(x)$ densities peaking more sharply at $x = 0$, which correspond to larger values of α . The performance of a classifier trained under a given hypothesis for the nuisance parameter (say, $\alpha = 1.5$) is then liable to be under- or over-estimated if the value of α is uncertain; the choice of a critical region $x > x^*$ corresponding

to a pre-defined FPR will similarly be affected, as will the resulting value of TPR.

In the given example the fraction of data selected in the critical region plays the role of our summary statistic, as we have assumed that it is the only input to a subsequent extraction of signal fraction. The fraction is of course affected by the unknown value of the nuisance parameter α , yet its value alone does not retain information about it: the statistic is therefore not sufficient. A sufficient statistic in this example would be the whole distribution of the classifier output shown by the observed data; that choice however fails to reduce, as desired, the dimensionality of the input, so it is not an effective summary for the inference task.

To discuss how an optimal choice of x^* based on the above densities may be influenced by the presence of the nuisance parameter α , we may consider the figure of merit called *approximate median significance* (AMS) [8], already introduced in Chapter 2:

$$\text{AMS} = \sqrt{2 \cdot \left[(N_s + N_b + N_r) \ln \left(1 + \frac{N_s}{N_b + N_r} \right) - N_s \right]}. \quad (13)$$

The AMS is a robust predictor of the statistical significance of an excess of observed events if a signal of mean rate N_s contributes to a data sample assumed to be only composed of background events coming from a Poisson distribution of known mean N_b ; the regularization term N_r reduces the impact of Poisson fluctuations in low-event-counts situations, preventing divergent behavior when N_b gets too low. If we set, e.g. $N_r = 10$ and compute the AMS as a function of the selection cut x^* for the three considered values of α of our toy model, and for a choice of $N_s = 20$, $N_b = 400$ in the data sample, we obtain the curves shown in the right panel of Fig. 1. It can then be observed, as expected, that the value of α affects both the peak value of the figure of merit and the optimal value of x^* which achieves it.

The above toy model exemplifies how not only do nuisance parameters have the power to modify the optimal working point of a ROC curve, but they also in general affect the overall classification performance, as well as the relative merit of different classifiers. For that reason, standard supervised classification techniques may not reach

optimality unless they more broadly address the conditionality issue stated above, or prove to be decoupled from it.^c

From a statistical point of view, in real-life situations “all models are wrong”, hence strictly speaking sufficient statistics that model the data may not exist! However, in most experimental situations approximate sufficiency is achievable, provided that the relevant nuisance parameters are included in the model and considered in the construction of the statistic. A number of brilliant ideas have been recently proposed to achieve that goal, in some cases effectively exploiting methods developed in Computer Science to endow learning algorithms of *domain adaptation* capabilities — the flexibility to achieve good results on data coming from one domain when trained with data coming from a different one, such as, e.g., the capability of driving a truck when trained to drive a car. In the context of point estimation in experimental particle physics, the different domains may involve a different relative importance of some of the features, the absence of others, or imperfections in the training model.

The growing interest in the development of new techniques to reduce or remove the effect of nuisance parameters in physics inference, powered by the availability of new machine learning tools and larger computing resources, has brought the focus of experimentalists on the central problem of how to achieve a true end-to-end optimization of physics measurements, and highlighted the need to pay undivided attention to the expected total uncertainty on the parameters of interest already in the training phase of learning algorithms. Below we provide an overview of methods developed to address those issues, and discuss their merits, applicability, and potential extensions.

^cIt must be noted here that a possible misalignment between the specification of the classification task and the true objective of the analysis should always be considered. In the given example we studied the AMS score as a robust proxy of the significance of an excess of signal events: such is a good choice when the objective of the analysts is the discovery of a yet hypothetical signal in the data. If, however, their focus were rather the setting of the most stringent upper limit on the signal rate — a common situation when the *a priori* sensitivity of a search does not offer chances of a discovery — then the whole learning task and optimization criteria would have to be revised.

2. Nuisance-Parameterized Models

A direct way to account for the effect of nuisance parameters in the construction of a summary statistic is to include them in the physical model through a parameterization of their effect on the observable event features. This requires the injection in the model of knowledge of their PDF from an external prior, or from an ancillary measurement, and may or may not be practical to implement depending on the problem.

In the simplest situations, as, e.g., when the problem is low-dimensional, a fully analytical solution may be sought. An example is offered by the decorrelation of the “ N -subjettiness ratio” variable τ_{21} [9] designed in the context of searches for the two-body decay of boosted resonances to reveal the resulting sub-structure in the produced hadronic jets. The variable may be likened to a classification score as it possesses large discrimination power against QCD background jets, but a selection based on its value biases the distribution of reconstructed “soft-drop” mass successively used for inference, because of its dependence on jet p_T , which can be here seen as a nuisance variable. As shown in [10], an analytical parameterization of the dependence of τ_{21} on jet p_T removes almost entirely the distortions in the mass spectrum. In the same context of boosted decay searches, similar results have been obtained for the observable D_2 [11].

In cases where experimental data are informative of the value of nuisance parameters, one may try to exploit that dependence in the construction of estimators for the parameters of interest. This situation was considered by Neal [12], who addressed the question of how the extraction of a sufficient summary for the signal fraction θ with a binary classifier is affected by unknown parameters α , when these modify the PDF of signal $p_s(x, \alpha)$ and background events $p_b(x, \alpha)$. The likelihood for N observations x_i ,

$$L(\theta, \alpha) = \prod_{i=1}^N [\theta p_s(x_i, \alpha) + (1 - \theta) p_b(x_i, \alpha)], \quad (14)$$

may be rewritten as

$$L(\theta, \alpha) = \left[\prod_{i=1}^N p_b(x_i, \alpha) \right] \cdot \prod_{i=1}^N \left[\theta \frac{p_s(x_i, \alpha)}{p_b(x_i, \alpha)} + (1 - \theta) \right]. \quad (15)$$

The first term in the right-hand side of the last expression is not a constant when nuisance parameters are present: factoring it out of the likelihood would therefore cause loss of information, since background events alone carry constraining power on the value of α . The usual classifier task of learning the ratio of signal and background PDFs $p_s(x)/p_b(x)$ is then no longer sufficient to solve the problem as it would be if no nuisances were present. The solution outlined in [12] involves the construction of low-dimensional summaries for both the nuisance parameters α and the observable event features x , using, e.g., a neural network. If good parametric models of the summaries can be constructed one may use them for inference, exploiting the informative power of the data themselves to constrain nuisance parameters. Approximate sufficiency can in principle be obtained with this recipe, if the parameterizations do not cause significant loss of information.

In other cases of HEP interest no knowledge or constraints on a nuisance parameter may be available, yet a parameterization of its effect on the observations successfully solves the issue. The classical example of this situation is the search for a new particle whose mass M_{true} is unknown, when signal events exhibit smooth variations in the momenta of the decay products as M_{true} changes.^d A classifier trained to distinguish the new particle from backgrounds using signal events simulated assuming a mass $M_1 = M_{\text{true}} + \alpha$ will consequently suffer a progressive degradation in performance as $|\alpha|$ increases. This was a common situation in early applications of binary classification to new particle searches, which focused on a mass range of particular interest, $M_{\text{true}} \simeq M_1$, and accepted the residual loss of power resulting for $\alpha \neq 0$. A more performant, yet CPU-consuming, solution [13, 14] was to independently train a set of classifiers C_i using, in turn,

^dA dependence of the particle branching fractions on M_{true} does not complicate matters if the resulting acceptance variations are known.

data simulated assuming different mass values M_i for the unknown signal. This approach is still sub-optimal in a general sense, since it does not fully exploit available resources (the simulated data). Each classifier is ignorant about the information processed by the other ones, as it only knows the precise mass hypothesis it corresponds to: in general, it is not possible to interpolate the results of different hypotheses.

A way to avoid the above shortcomings, first proposed in [15], is to parameterize the effect of the nuisance parameter in the construction of the classifier. This may be achieved by including the unknown value of M_{true} within the set of features that describe simulated signal events; for background events an arbitrary mass value, or one chosen at random for each different training event, is correspondingly added. A suitable admixture of training data with signal events corresponding to different M_i hypotheses spanning the range of interest may then be used in the learning phase.^e The benefit of this procedure is that it yields an interpolated classification score C_k even for events with mass values M_k never seen during training. With this approach, a smoother dependence is often expected if the classifier is a neural network rather than, e.g., a decision tree.

The effectiveness of this strategy was demonstrated [15] using a simple neural network (NN) architecture for the discrimination of a new particle X decaying to top–antitop quark pairs from non-resonant $t\bar{t}$ backgrounds in proton-proton collision data reconstructed by the ATLAS detector as simulated by DELPHES [16]. It was shown how for a given specific mass hypothesis M_1 a parameterized NN performed similarly to a non-parameterized NN trained with signal at the same mass, but it outperformed it for all other masses (red curve in Fig. 2), even when the non-parameterized NN was trained with an admixture of mass values (black curve).

The parameterization of the dependence of observable variables on the latent features of the underlying physical model — which

^eRegardless of the *a priori* choice of signal admixture employed in the training, the resulting inference cannot be considered Bayesian, as the choice only affects the power of the classifier.

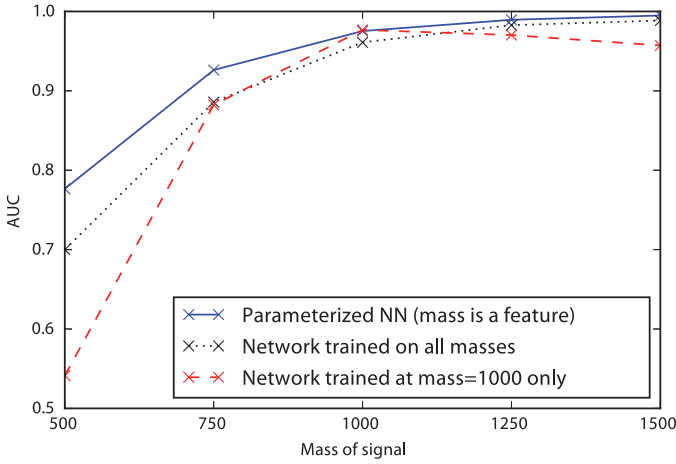


Fig. 2. Area under the ROC curve for binary classification in the search for $X \rightarrow t\bar{t}$ on simulated ATLAS data as a function of the particle mass M_i of test data. The parameterized NN (blue line) outperforms non-parameterized NN trained at a single mass value ($M_i = 1000$ GeV, red dashed line) or trained with a mixture of signal samples for different M_i values (black dotted line). Reprinted with permission from [15].

include both interesting and nuisance parameters — has been more generally considered [17] in the context of likelihood-free inference. The proposed algorithm performs a dimensionality reduction of the data through a parameterization that is monotonic with the likelihood ratio, allowing optimal inference via a calibration of the output of a binary classifier. We refer the reader to Sec. 5.2 for more detail on this approach.

3. Feature Decorrelation, Penalized Methods, and Adversary Losses

When a direct parameterization of the effect of nuisance parameters on the summary statistic used for classification proves ineffective or impractical to implement, there are several possible alternatives. In a few specific applications it is sufficient to operate a suitable preprocessing of training data that reduces or removes the dependence of the classifier output on a variable sensitive to nuisance parameters.

A second class of solutions aim to make the classifier score insensitive to variations in the value of nuisances by engineering a robust optimization objective for the classification task. Finally, a more radical approach is to change the overall architecture of the algorithm used in the search of the optimal solution, using adversarial techniques to find the best compromise between signal discrimination and impact of nuisances. Below we briefly discuss each of these approaches.

3.1. *Mass decorrelation*

The intensive search for new physics carried out by the ATLAS and CMS experiments in final states dominated by QCD backgrounds fostered, in the past decade, the development of a number of imaginative new methods to increase signal purity without modifying the shape of the distribution of reconstructed mass, M_{rec} , of the hypothetical new particle, which is commonly used at the end of the selection step to estimate or limit the signal contamination in the sample. Since the QCD background is complex to model reliably, a selection cut on the output of a well-trained classifier does not guarantee optimal inference on the presence of any signal, because the background retained by the cut is usually biased toward displaying a “signal-like” mass distribution. In this situation M_{rec} is not in itself a nuisance parameter; however, the reduction of its discrimination power caused by the selection enhances the impact of background normalization and shape uncertainties on the estimate of signal fraction. Further, a reshaping of the background distribution complicates the application of bump hunting techniques, for example by hindering the use of data-driven background estimates based on mass sidebands.

The most straightforward way to reduce the dependence of a classification score on M_{rec} (or any other specific observable of relevance for inference downstream of the selection) is called “planing” [18, 19]. A simple way to implement planing is to pre-select training samples for signal and background such that they have the same marginal PDF in the variable one aims to decorrelate, $p_S(M_{\text{rec}})^{\text{sel}} = p_B(M_{\text{rec}})^{\text{sel}}$. As the above corresponds to making limited use of available training data, it proves more effective to weight

each event i by a mass-dependent value $w(M_{\text{rec},i})$ derived from the PDFs of the two training datasets, $p_S(M_{\text{rec}})^{\text{train}}$ and $p_B(M_{\text{rec}})^{\text{train}}$,

$$w(M_{\text{rec},i}) = \begin{cases} 1/p_S(M_{\text{rec},i})^{\text{train}}, & i \in S \\ 1/p_B(M_{\text{rec},i})^{\text{train}}, & i \in B \end{cases}. \quad (16)$$

The weights $w(M_{\text{rec},i})$ enter directly the calculation of the loss function (e.g., binary cross-entropy) of the classifier in the training stage, but are not used for validation and testing. Planing has been shown to significantly reduce the dependence of classifier output on the planed variable in specific situations, and due to the simplicity of its implementation it may constitute a quite practical solution to the problem; however, its effectiveness is limited when other event features in one or both classes indirectly inform the classifier on the value of the planed variable, if the latter — as is often the case — carries discriminant power. In the context of searches for new physics in boosted hadronic jets, a decorrelation of the output of a NN classifier from the mass of the boosted jet was instead achieved by feature preprocessing based on principal component analysis [18]. The proposed method involved the PCA rotation and standardization of 17 employed NN inputs (a basis set of N -subjettiness variables τ_N^β proposed in [20]) from trained data suitably binned in jet mass. Besides avoiding the sculpting of the jet mass distribution of QCD background events, the resulting classifier was shown to be effective for signal discrimination also at signal masses for which it was not trained.

3.2. *Modified boosting and penalized loss methods*

As mentioned above, a decorrelation of the classifier output from a variable of interest x may be difficult to obtain with data preprocessing techniques when other event features are informative of the value of x , especially if x itself contains discriminant information.

The search for new low-mass resonances in Dalitz plots [21] or with amplitude analysis provides strong motivation to achieve uniformity of a classifier selection as a function of kinematical variables of interest, as systematic uncertainties may be greatly amplified by

the unevenness of selection efficiency. The first algorithm developed to explicitly target that use case was *uBoost* [22], which relies on boosted decision trees to improve signal purity. The method builds on the standard AdaBoost prescription [23] of increasing the weight of training events misclassified by the decision tree built in the previous iteration of the BDT sequence, augmenting it by modifying the weight of signal events depending on the disuniformity of the selection. If w_i^{n-1} is the weight of event i at boosting iteration $n-1$, the new weight is computed as

$$w_i^n = c_i^n u_i^n w_i^{n-1}, \quad (17)$$

where $c_i^n = \exp(-\gamma_i p_i^{n-1})$ is the AdaBoost classification weight, with $\gamma_i = +1$ (-1) for signal (background) events and where p_i is the prediction of the previous decision tree in the series.

The uniformity weight u_i^n is defined as the inverse of the density of signal in the proximity of event i , and is computed with the k-nearest-neighbor algorithm; for background events u_i is set to unity. Since it is necessary to consider many different values of signal efficiency in the construction of the final BDT score and to the use of kNN, the CPU cost of training with *uBoost* is higher than that of a regular BDT, although not prohibitive in practical applications. Tested on a Dalitz analysis, the method was shown to achieve the wanted uniformity with almost no loss in classification performance [22].

Following on the thread of *uBoost*, a number of interesting alternatives to achieve uniform selection efficiency of a BDT classifier were introduced in [24], again targeting the use case of Dalitz plot analysis. The algorithm called kNNAdaBoost achieves the uniformity goal by modifying the AdaBoost weights to include information on the classification probability of k nearest neighbors to each event,

$$w_i^n = w_i^{n-1} \exp \left[-\gamma_i \sum_j a_{ij} p_j \right], \quad (18)$$

where the a_{ij} matrix collects information on the density of events of the same class around event i , by setting $a_{ij} = 1/k$ if j is among the k neighbors of i and $= 0$ otherwise. Other methods proposed in [24]

involve the use of a_{ij} in the loss of the classifier, minimized with the use of gradient boosting. These techniques are shown to improve over uBoost by achieving better uniformity in specific use cases.

More recently, the issue of decorrelation from variables of interest or, more generally, robustness to nuisance parameters has been addressed by using neural network classifiers, adding suitable regularizer terms to their loss function. An option discussed in [25] is to use, for that purpose, a measure of the extent to which two sets of features \vec{x} , \vec{y} are independent.^f The proposed measure is dubbed *DisCo* ("distance correlation") [25], a function of the considered features which can be constructed by first defining a distance covariance

$$\begin{aligned} \text{dCov}^2(X, Y) = & \langle |X - X'| |Y - Y'| \rangle + \langle |X - X'| \rangle \langle |Y - Y'| \rangle \\ & - 2 \langle |X - X'| |Y - Y''| \rangle, \end{aligned} \quad (19)$$

where $|\cdot|$ is the Euclidean vector norm and (X, Y) , (X', Y') and (X'', Y'') are i.i.d. pairs from the joint distribution of the two features; brackets indicate taking averages. The distance correlation, defined as

$$\text{dCorr}^2(X, Y) = \frac{\text{dCov}^2(X, Y)}{\text{dCov}(X, X) \text{dCov}(Y, Y)} \quad (20)$$

is then bound between 0 and 1, and is null only if x and y are fully independent. $\text{dCorr}^2(X, Y)$ is differentiable and can be computed from batches of data samples; its value can be profitably added as a penalty term to the loss of the classifier, once multiplied by a positive hyperparameter λ controlling its strength. The multiplier allows to gain control over the acceptable amount of interdependence of x and y achieved by a minimization of the penalized loss. In the single-dimensional application considered in [25] DisCo proves competitive or advantageous over, e.g., adversarial setups (see below) or other methods; further studies are needed to gauge its performance in more complex situations.

^fIn their work, Kasiyczka and Shih discuss the classical single-dimensional case when $x = M_{\text{rec}}$ is the mass of a searched particle and $y = c$ the output of the classifier itself, but the extension to multi-dimensional problems is straightforward.

A similar approach is taken [26] in a study more explicitly aiming at a reduction of the dependence of classifier score from nuisance parameters α . In the proposed technique the n -bin histogrammed distribution \mathcal{N}_k of classifier output $f(x)$ from input features x is first made differentiable with the use of a Gaussian smoothing with functions \mathcal{G}_k ,

$$\mathcal{N}_k(f(x)) = \sum_b \mathcal{G}_k(f(x)), \quad (21)$$

where k runs on the bins and b runs on the training events in a batch. The usual loss L_0 of the classifier can then be penalized by a term derived from the difference in smoothed bin counts of the original output $f(x)$ and its nuisance-varied value $f(x + \alpha)$,

$$L(\lambda) = L_0 + \lambda \frac{1}{n} \sum_k \left(\frac{\mathcal{N}_k(f(x)) - \mathcal{N}_k(f(x + \alpha))}{\mathcal{N}_k(f(x))} \right)^2. \quad (22)$$

The modified loss effectively decouples the classifier output from the value of α , both in a synthetic example and in the benchmark problem of $H \rightarrow \tau\tau$ discrimination proposed in the ATLAS kaggle challenge [27], where the τ lepton momentum scale is considered as the nuisance parameter.

3.3. Adversarial setups

The construction of an adversarial setup where two independent neural networks are pitched one against the other in the search for the optimal working point in a constrained classification problem may be considered an extension, if not the logical next step, of the penalized loss methods discussed above. In fact, the global loss function is still the combination of two parts, one of which is the usual classification loss (e.g., a BCE term) and the other is a penalization contributed by the adversary, usually modulated by a regularization multiplier λ . The difference is that adversarial architectures create a conceptual symmetry between the classification task aiming at a signal-background separation and the discrimination of different values of a nuisance parameter, putting the two minimization problems on equal footing.

The idea of using a classifier trained to discriminate between data from different domains to constrain the error of a binary classifier trained in one domain and applied to a different domain dates back to early computer science research [28, 29]. These works provide foundations to the more modern domain adversarial approaches which formulate the domain adaptation approach through a min-max learning objective. Applications in HEP arise when training and test data come from different domains (a source and a target one), or when training data are simulated by an imperfect model of real (test) data. It was shown that robust classification can be achieved in such situations if one can find a suitable representation of the data which is maximally insensitive to their source. An adversarial neural network is thus tasked to learn such a representation while competing with one that tries to achieve maximal separation of labeled classes of training data [30].

The first proposal of adversarial neural networks to achieve robustness to systematic uncertainties in HEP problems was the one of Louppe, Kagan and Cranmer [31], who showed the feasibility of using adversarial techniques to make the classification score $f(X; \theta_f)$ a “pivotal quantity” in the statistical sense [32], i.e. one whose distribution is independent on the value of nuisance parameters z ; above, θ_f are the parameters of the classifier, and X denote the data. If one further denotes the adversary, r , with parameters θ_r , whose task is to discern values of z from the output value $f(X, \theta_f)$ of the classifier, the loss functions of the two networks may be succinctly written $\mathcal{L}_f(\theta_f)$ and $\mathcal{L}_r(\theta_f, \theta_r)$, and a simultaneous training can be carried out by using the value function

$$E(\theta_f, \theta_r) = \mathcal{L}_f(\theta_f) - \mathcal{L}_r(\theta_f, \theta_r) \quad (23)$$

which can be optimized by the minimax solution

$$\hat{\theta}_f, \hat{\theta}_r = \arg \min_{\theta_f} \max_{\theta_r} E(\theta_f, \theta_r). \quad (24)$$

Convergence to the optimal solution cannot be guaranteed if the nuisance parameters shape the decision boundary directly. In that case a hyperparameter λ multiplying the adversary loss \mathcal{L}_r may be introduced in Eq. (23), and a search for approximate optimality must

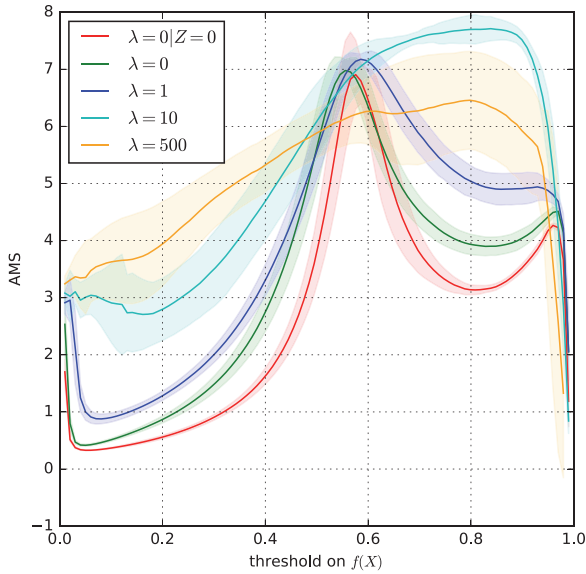


Fig. 3. AMS score as a function of classifier score for a binary classification task, for different values of the hyperparameter λ modulating the loss penalization, and for the case when no nuisance parameter is present. For $\lambda = 10$ an advantageous tradeoff of classification accuracy and robustness to the nuisance is obtained at high classification scores. Reprinted with permission from [31].

be performed. As an example, Louppe *et al.* consider both a synthetic example and a HEP use case when the nuisance parameter Z is categorical, describing the absence ($Z = 0$) or presence ($Z = 1$) of pile-up in LHC collisions data. In the latter case they show (see Fig. 3) how an effective compromise between the classification and the pivotal tasks may be found by a tuning of λ .

The application of the above technique to the discrimination of the decay of boosted heavy particles in a situation where background systematics affect the inferential step downstream of the NN-based selection was considered in [33]. In their work, authors showed how the relevant utility function in the problem — the significance of a resonant signal in the data, once systematic uncertainties were accounted for — was indeed maximized by an adversarially trained classifier, despite its slight degradation of separation power with respect to a non-adversarial classifier.

A further comparison of the effectiveness of the adversarial training proposed in [31] to alternatives based on data augmentation and tangent propagation, for the goal of optimizing classification in presence of nuisance parameters, was produced in [34]. The considered HEP problem was the one of $H \rightarrow \tau\tau$ discrimination from backgrounds proposed by the Higgs Kaggle Challenge [27], where an uncertainty on the τ lepton energy scale was introduced and propagated to the input features of signal and background. In addition to a baseline, non-systematics-aware neural network classifier, they employed in their comparison a data augmentation method based on training datasets constructed so as to appropriately sample the relevant range of values of the nuisance parameter. Finally, the tangent propagation method consisted in modeling nuisance parameters as coherent geometric transforms of the event features, operated by differentiable functions; a regularization of the model was provided through the derivative of the classifier score on the nuisance parameter value, as introduced by Simard *et al.* [35]. The comparison showed that adversarial learning had a minor advantage over data augmentation, although further work was deemed necessary to achieve more conclusive results on the matter. Tangent propagation was instead shown to be unsuccessful on the specific problem considered, due to large uncertainties introduced in the geometrical transformation caused by the large class overlap in the feature space.

We conclude this survey of applications of adversarial techniques to constrain the effect of nuisance parameters with a mention of two very recent studies. The first, by Blance, Spannowsky, and Waite [36], examines adversarial classification as a preliminary step to the use of autoencoders for unsupervised classification, to verify their effectiveness in reducing the dependence of the autoencoder task on systematic uncertainties. They apply this idea to the search of resonances decaying to semileptonic $t\bar{t}$ final states, showing independence of the resulting classification task on the considered smearings of the input models. A second interesting study [37] attacks the problem of theoretical uncertainties with adversarial networks. Since uncertain theory parameters affect the data in a coherent way, they can be controlled more effectively than experimental ones in

machine learning applications. Authors consider the case of searches for new physics in events with a Higgs boson and a high-momentum jet, where renormalization and factorization scale variations heavily affect the predictions of standard model backgrounds, making traditional discrimination methods unreliable. Sensitivity to new physics can be retained by an adversarial technique which ensures robustness to theoretical scale uncertainties, at the price of smaller discrimination power.

Overall, adversarial methods discussed in this section prove effective to achieve approximate independence of the classification from the value of selected input features. In general, however, there is no guarantee that the resulting equilibrium point between the two competing tasks be optimal for the final goal of the analysis in which they are embedded. For this reason, the hyperparameter λ governing the tradeoff between the two losses must be optimized independently. More direct ways to strive for a complete optimization of classification in physics measurements and searches are examined in Sec. 5.

4. Semi-supervised Approaches

At the beginning of this chapter, nuisance parameters were introduced as additional parameters that account for the limitations of the description of the data and have to be considered when making accurate statistical statements. Given that most machine learning models in HEP are usually trained using simulated observations, the resulting models could only aspire to be optimal at the task at hand (typically, classification or regression) for the specific configuration of nuisance parameters used for data generation. The previous sections discussed some solutions to this problem, such as parameterizing the model or decorrelating its output using additional loss terms. In this section we review alternative approaches that are based on using actual experimental data to complement or substitute simulated samples in the model training procedure, focusing on how these techniques could help to deal with nuisance parameters.

Experimental data are the source of information used to test hypothesis or estimate parameters given a model. Models are usually

based on detailed simulations of the underlying physical processes and the detector response, providing, in general, a quite good although not perfect description of the data. Oftentimes, experimental data from well-known processes are also used to cross-check the accuracy of the description by the model and to estimate correction factors and associated uncertainties as necessary. These calibration procedures, which also constitute statistical inference analyses in their own right, provide a mechanism to reduce mismodeling issues and obtain data-based estimates for nuisance parameter constraints. While general calibrations are typically performed experiment-wide, more detailed calibrations are often carried out for specific analysis scenarios to improve their precision and discovery reach, for example using an independent subset of data that is expected to be well-modeled to further correct or constrain known unknowns at the inference stage. In some cases, yet arguably not often in analyses that use machine learning to reduce the dimensionality of their summary statistics, known properties of experimental data allow us to use a well-understood subset to model one of the mixture components.

The interrelation between experimental data and the generative model and its parameters in HEP is thus more involved than its ideal depiction in statistical literature. When training supervised machine learning models using simulated data, the expected performance at the objective task in experimental data improve if training and validation datasets are well calibrated and correspond to the best estimates of the parameters of interest.^g Leaving aside the issue of whether the supervised learning task is a good proxy of the analysis inference goal when nuisance parameters are important, which will be discussed in Sec. 5, we review here methodologies that use experimental data during training to close the gap between the performance at the inference task between real and simulated data.

^gFor completeness, we note that even when the machine learning model is not trained with the most accurate description of the data, it is still possible to make calibrated statistical statements, as long as known unknowns are properly accounted for in the statistical model used for inference.

Many of the efforts to achieve the above goal are based on innovations from *weak supervision* and *semi-supervised learning*, that focus on the problem of learning useful models from partial, non-standard, or noisy label information. In this context, when considering a classification task, simulated observations can be considered as fully labeled data that provide a possibly imperfect description, while real data observations can be thought as unlabeled or very sparsely labeled mixtures from different classes, which however do not suffer from the same imperfection. For example, Dery *et al.* [38] propose an approach based on *learning from label proportions* (LLP), where a neural network is trained using a custom loss that is only based on the known class proportions for given sets of training data. They validate the method on a quark vs. gluon tagging example problem, finding that it can be used to obtain a similar performance to that of a fully supervised classifier, while being more robust to simulation mismodeling of the input variables.

One of the potential advantages of approaches based on learning from label proportions (and weak supervision more generally) is that in principle they could be extended to train the classifier directly using data from the experiment. However, the LLP approach requires at least knowledge of the label proportions in the mixed samples, which might not be available at training time. To address this limitation, Metodiev *et al.* [39] propose a new paradigm referred to as classification without labels (CWoLA), where the classifier is trained to distinguish between two mixed sample with different (and possibly unknown) component fractions. This also simplifies the previous approach because it is based on standard classification loss, where the label is not the observation class but an identifier of the mixed sample the event belongs to, as depicted in Fig. 4. Authors prove that the optimal binary classifier (in the Bayes sense) for distinguishing events from the mixed samples is a simple function of the density ratio between the components. Furthermore, they demonstrate that CWoLA as well as LLP perform similarly to a fully supervised classifier on pure samples, using practical examples such as a quark/gluon discrimination problem.

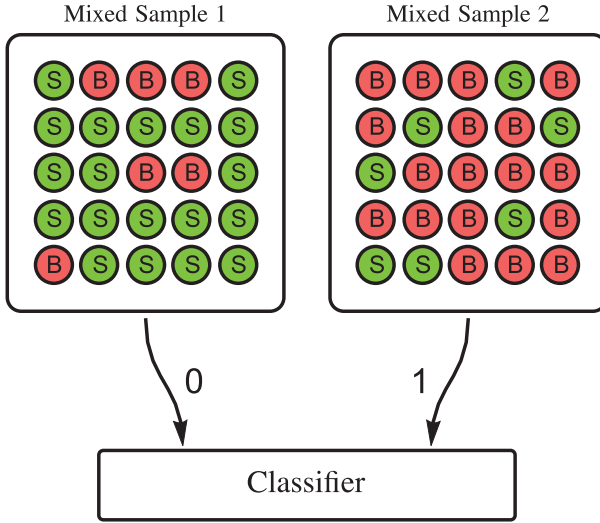


Fig. 4. An illustration of the CWoLa framework. Figure and description by the authors of CWoLa [39] and licensed under CC BY 4.0.

While CWoLa has a wider range of applicability than standard LLP techniques, it also requires two (possibly smaller) mixed test data samples with known fractions to establish operating points. After its introduction, two other studies have applied variations of CWoLa to sample use cases. Cohen *et al.* [40] applied weakly supervised neural networks to the search of gluino production using fast simulation samples, demonstrating that weak supervision can perform similarly to full supervision and that it is robust to certain types of mismodeling. Further work by Komiske *et al.* [41] has shown that weak supervision approaches scale well to problems with high-dimensional inputs and larger models, by successfully applying LLP and CWoLa to the quark/gluon discrimination problem using a convolutional network model applied directly to jet images.

The most attractive feature of weak supervision techniques such as LLP and CWoLa is that, in principle, they may enable the use of real data during the training procedure. The use of experimental data for training with this family of techniques has however not been demonstrated in HEP practice so far. In the best case scenario,

a weakly supervised classifier trained with data could be used to extract the optimal classifier (in the Bayes sense) between each of the mixture components (e.g., signal and background). The output of this classifier could then be used to select or construct a summary statistic to carry out the inference goal of the analysis. Yet most likely, the model would have to be constructed using simulated observations that are subjected to the effect of nuisance parameters. Hence a potentially Bayes-optimal weakly supervised classifier would suffer the same pitfalls as any other classifier in relation with the analysis inference goal, as we discuss below in Sec. 5.

Additionally, we note that if experimental data are used during training the model might be overfitted to the particular statistical fluctuations of the dataset, so an experimental data splitting scheme, or the use of experimental data from an independent subset, might be needed to avoid biased estimations. If the data representing different mixture fractions are taken from different control regions, the previous caveat may be avoided, however the density of the components for each of the mixed samples would then not necessarily be the same, invalidating the basic theoretical assumption of CWOLA or LLP. The fundamental assumption also does not hold if the distributions in the control region exhibit different correlations than in the signal region.

In conclusion, while weak supervision could be useful to build classifiers that might benefit the model classification performance, due to their being more robust to certain types of mismodeling, existing practical approaches do not fully address the issue of dealing with nuisance parameters.

5. Inference-Aware Approaches

The approaches discussed so far use diverse methodologies in order to overcome situations where the data generating process is not perfectly known and thus the performance of the supervised learning task considered (typically classification) might be degraded once it is applied on real data. However, recent work has shown that some of

the innovations in the field of machine learning are flexible enough so as to be re-purposed to deal more closely with the statistical inference objective of HEP analyses.

The solutions discussed in this section move away from the overall goal of optimizing models to become performant at proxy supervised learning tasks such as classification, and attempt to frame the problem directly as one of statistical inference. This change of paradigm is often referred to as likelihood-free or simulation-based inference, and is a rapidly evolving line of research, with applications within particle physics as well as in other scientific domains that heavily rely on complex generative models, such as epidemiology or cosmology.

For a broader overview of the techniques proposed to deal with this problem and their role in particle physics we refer to Chapter 16 and other general reviews [42]. In this section, we instead focus on how some of these inference-aware approaches could be useful to deal with nuisance parameters in the context of particle physics. Given that most of these solutions already cast the problem in the form of statistical inference on a set of parameters given the data, it is not surprising that they allow nuisance parameters to be incorporated or dealt with in a principled way.

5.1. *Why are classification and regression not enough?*

Before delving into these new techniques, it is worth considering the limitations of classification and regression as proxy supervised tasks from the point of view of statistical inference. For simplicity, let us consider the paradigmatic problem of inference about the mixture coefficient in a two-component mixture model, which is often the basis of cross section measurements or searches for new physics processes:

$$p(\mathbf{x}|\mu, \boldsymbol{\theta}) = (1 - \mu)p_b(\mathbf{x}|\boldsymbol{\theta}) + \mu p_s(\mathbf{x}|\boldsymbol{\theta}) \quad (25)$$

where μ is a parameter corresponding to the signal mixture fraction, \mathbf{x} is the event feature space and $\boldsymbol{\theta}$ are other parameters which the

component distribution functions might depend on. For the problems of relevance to machine learning techniques, we may assume that the probability density functions for signal $p_s(\mathbf{x}|\boldsymbol{\theta})$ and background $p_b(\mathbf{x}|\boldsymbol{\theta})$ are not known parametrically, yet we have access to random samples from a simulator that is able to model them implicitly.

The relation between the density ratio approximations from Eq. (1) and the typical problems of inference in HEP may be studied using two different statistical constructions: likelihood ratios or summary statistics. Both approaches lead to equivalent conclusions regarding the limitations of classification as a means of obtaining useful transformations for statistical inference in the presence of nuisance parameters, but they are both relevant because they imply ways of framing the problem which map very well to different families of new techniques built to address this issue which we discuss later in this section.

Let us start with likelihood ratios, which can be generally defined for a set of n data observations $D = \{\mathbf{x}_0, \dots, \mathbf{x}_n\}$ between two simple hypotheses H_0 and H_1 as:

$$\Lambda(D; H_0, H_1) = \frac{p(D|H_0)}{p(D|H_1)} = \prod_{\mathbf{x} \in D} \frac{p(\mathbf{x}|H_0)}{p(\mathbf{x}|H_1)}, \quad (26)$$

where the last expansion requires independence between observations, and where we note that the quantity $p(\mathbf{x}|H_0)/p(\mathbf{x}|H_1)$ is a density ratio and could be approximated as discussed before by training a probabilistic classifier to distinguish samples generated under each hypothesis. From the Neyman–Pearson lemma [43], we know the likelihood ratio is the most powerful test statistic to distinguish the two simple hypotheses H_0 and H_1 at given significance level $\alpha = P(\Lambda(D; H_0, H_1) \leq t_{\text{cut}})$, for any threshold t_{cut} .

Going back to problems where hypotheses have a mixture structure like the one discussed in Eq. (25) and differ in their mixture composition, this would mean training a classifier using samples generated from $p(\mathbf{x}|\mu, \boldsymbol{\theta})$ for the specific mixture fractions μ_0 and μ_1 that characterize each of the hypothesis $p(\mathbf{x}|H_0) = p(\mathbf{x}|\mu_0, \boldsymbol{\theta})$ and

$p(\mathbf{x}|H_1) = p(\mathbf{x}|\mu_1, \boldsymbol{\theta})$. This would rapidly become cumbersome if we were dealing with multiple tests for a set of different μ_0 and μ_1 values. Luckily, each factor in the likelihood ratio from Eq. (26) can be expressed in the following manner:

$$\frac{p(\mathbf{x}|H_0)}{p(\mathbf{x}|H_1)} = \frac{(1 - \mu_0)p_b(\mathbf{x}|\boldsymbol{\theta}) + \mu_0 p_s(\mathbf{x}|\boldsymbol{\theta})}{(1 - \mu_1)p_b(\mathbf{x}|\boldsymbol{\theta}) + \mu_1 p_s(\mathbf{x}|\boldsymbol{\theta})} \quad (27)$$

$$\begin{aligned} &= \left(\frac{1 - \mu_1}{1 - \mu_0} + \frac{\mu_1}{1 - \mu_0} \frac{p_s(\mathbf{x}|\boldsymbol{\theta})}{p_b(\mathbf{x}|\boldsymbol{\theta})} \right)^{-1} \\ &\quad + \left(\frac{1 - \mu_1}{\mu_0} \left(\frac{p_s(\mathbf{x}|\boldsymbol{\theta})}{p_b(\mathbf{x}|\boldsymbol{\theta})} \right)^{-1} + \frac{\mu_1}{\mu_0} \right)^{-1} \end{aligned} \quad (28)$$

so for a given pair μ_0 and μ_1 the density ratio between hypotheses in the likelihood ratio is a bijective function of the ratio $p_s(\mathbf{x}|\boldsymbol{\theta})/p_b(\mathbf{x}|\boldsymbol{\theta})$. The latter quantity can be approximated by training a probabilistic classifier to distinguish signal and background simulated samples, which is computationally more efficient and easier to interpret intuitively than the likelihood ratio itself.

A likelihood ratio approximation can thus be obtained in the case of two simple two-component mixture hypotheses that only differ in the mixture fractions by plugging the output of a probabilistic classifier $c(\mathbf{x})$ trained to distinguish signal and background observations in Eq. (28) with the corresponding values of μ_0 and μ_1 in Eq. (26). Oftentimes, the calculation of the likelihood ratio is not necessary because the classifier output directly contains all the relevant information about the ratio approximation. Hence the classifier output can be used directly as a summary for inference with the help of histograms or non-parametric density estimation techniques, with the added advantage that it is typically a $[0, 1]$ bounded variable and thus easy to interpret. It is worth mentioning that the relation between the likelihood ratio and the density ratios of the pair of mixture components can also be useful in the multi-component mixture setting. In that case, the likelihood ratio factor can be expressed in terms of the density ratios that can be obtained for each pairwise component classification problems [17].

Within this framework, the usefulness of probabilistic classifiers that distinguish signal and background observations is that they can be used to approximate the likelihood ratio, which is the most powerful summary statistic for two simple hypothesis that differ only on the mixture fraction parameters. If the hypotheses are not fully specified, i.e. they depend on additional parameters (the dependence with the mixture fractions can be factored out as discussed before), the likelihood ratio as defined in Eq. (26) also depends on these parameters. The Neyman–Pearson lemma does not hold when parameters are varied nor for composite generalizations such as the profile likelihood ratio. Hence, when nuisance parameters are important, a fixed probabilistic classifier, even if optimal in the Bayes sense, is not guaranteed to provide a transformation that is optimal for inference in any statistically meaningful way.

An alternative formulation of the limitations of classification for statistical inference is based on the sufficiency conditions required for summary statistics, according to the Fisher–Neyman factorization criterion. A summary statistic for a set of i.i.d. observations $D = \{\mathbf{x}_0, \dots, \mathbf{x}_n\}$ is sufficient with respect to a statistical model and a set of parameters $\boldsymbol{\theta}$ if and only if the generating probability distribution function of the data $p(\mathbf{x}|\boldsymbol{\theta})$ can be factorized as follows:

$$p(\mathbf{x}|\boldsymbol{\theta}) = q(\mathbf{x})r(\mathbf{s}(\mathbf{x})|\boldsymbol{\theta}), \quad (29)$$

where $q(\mathbf{x})$ is a non-negative function that does not depend on the parameters and $r(\mathbf{x})$ is also a non-negative function for which the dependence on the parameters \mathbf{x} is a function of the summary statistic $\mathbf{s}(\mathbf{x})$. Such a sufficient statistic contains all the information in the observed sample useful for computing any estimate on the model parameters, and no complementary statistic of the observed data can add information about $\boldsymbol{\theta}$.

A trivial sufficient summary statistic according to the previous definition is the identity function $\mathbf{s}(\mathbf{x}) = \mathbf{x}$, yet typically we are only interested in summaries that reduce the original data dimensionality. If $p(\mathbf{x}|\boldsymbol{\theta})$ is not known in closed form, as is often the case in HEP analyses, the general task of finding a sufficient summary statistic that reduces the dimensionality cannot be tackled directly

by analytic means. An exception to this can be easily shown in the case of a mixture model where the mixture fraction μ is the only parameter. By both dividing and multiplying by the mixture distribution function from Eq. (25) we easily obtain:

$$p(\mathbf{x}|\mu, \boldsymbol{\theta}) = p_b(\mathbf{x}|\boldsymbol{\theta}) \left(1 - \mu + \mu \frac{p_s(\mathbf{x}|\boldsymbol{\theta})}{p_b(\mathbf{x}|\boldsymbol{\theta})} \right) \quad (30)$$

from which we can already prove that the density ratio $s_{s/b}(\mathbf{x}) = p_s(\mathbf{x}|\boldsymbol{\theta})/p_b(\mathbf{x}|\boldsymbol{\theta})$ (or alternatively its inverse) is a sufficient summary statistic for the mixture coefficient parameter, according to the Fisher–Neyman factorization criterion from Eq. (29). This quantity could be efficiently approximated by considering the problem of probabilistic classification between signal and background as discussed in Eq. (1). Because any bijective function of a sufficient summary statistic is also a sufficient summary statistic, the conditional probability from the conditional output of a balanced classifier

$$c(\mathbf{x}) = s_{s/(s+b)}(\mathbf{x}) = \frac{p_s(\mathbf{x}|\boldsymbol{\theta})}{p_s(\mathbf{x}|\boldsymbol{\theta}) + p_b(\mathbf{x}|\boldsymbol{\theta})} \quad (31)$$

can be used directly as a summary instead of $s_{s/b}(\mathbf{x})$, with the additional advantage that it is bounded between zero and one, a fact that greatly simplifies visualization and calibration.

From this perspective, the utility of signal vs. background classification to obtain an approximately sufficient summary statistic with respect to the mixture model and mixture fraction μ is evident. However, if the statistical model depends on additional nuisance parameters, even Bayes optimal probabilistic classification does not provide any sufficient guarantees. Thus, even for the best possible classifier that can be constructed, useful information which can be used to constrain the parameters of interest might be lost if a low-dimensional classification-based summary statistic is used in place of the original data \mathbf{x} .

Above we have reviewed from a statistical perspective the limitations of signal vs. background classification models when the goal is inference in the presence of nuisance parameters. In practice, classifiers can be trained for the most probable likely value of the nuisance

parameters and their effect can be adequately accounted for during calibration, yet the resulting inference will be degraded even if the classification is optimal. Alternative uses of classification and regression models such as particle identification and momentum or energy regression can be understood as approximations of a subset of relevant latent variables \mathbf{z} of the generative model. This information could be then be used to complement the reconstruction output for each object and design better hand-crafted or classification-based summary statistics, so at the end the final goal is inference, and the previously mentioned shortcomings still apply.

5.2. *Generalizing the likelihood ratio trick*

The first known attempts to study the relation between statistical inference in HEP with nuisance parameters and probabilistic classifiers were made by Neal [12]. In his seminal paper, in addition to making explicit the problem of not being able to compute the data generating likelihood in closed form, as well as clarifying the useful relation between likelihood ratios and probabilistic classifiers discussed in the previous subsection, he also acknowledges the limitations of this approach in the presence of nuisance parameters and suggests a few possible candidate solutions.

To ameliorate the problems of losing useful information when reducing the dimensionality of the data with summary statistics, a few variations over classical signal vs. background classifiers trained with the best estimation of the nuisance parameters are proposed. The first proposal foresees the training of a single robust classifier by combining simulated observations of signal and background generated with different values of nuisance parameters, for example drawn from their prior or from a reasonable distribution, to constrain the nuisance parameters $\pi(\boldsymbol{\theta})$.

The drawbacks of such marginal classifier are similar to the concerns about models trained for the most likely values of the nuisance parameters: it might not be possible to accurately classify without knowing $\boldsymbol{\theta}$, and even when that is possible the usefulness of the resulting score will be degraded when calibrated statistical inference is

carried out. To address these concerns, the author suggests a generalization based on training a single classifier considering both the observations \mathbf{x} and the nuisance parameters $\boldsymbol{\theta}$ as input. The resulting model would be a nuisance-parameterized signal vs. background classifier, thus an early precedent for some of the approaches discussed in Sec. 2. In order to use these parameterized classifiers on real data, for which the correct values of $\boldsymbol{\theta}$ are not known, Neal argues that an additional per-event regression model for $\boldsymbol{\theta}$ could be trained on simulated observations.

The ideas developed by Neal [12] were not applied in practice until they were generalized and extended by Cranmer *et al.* [17]. The authors of the latter work identify the same problem regarding the use of discriminative classifiers to approximate likelihood ratios with nuisance parameters, and introduce a generic framework for inference using calibrated parameterized classifiers referred to as CARL. In their more general formulation, they propose using a doubly parameterized classifier to approximate the likelihood ratio for all possible pairs of relevant parameters $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$ of a generative model $p(\mathbf{x}|\boldsymbol{\theta})$ as follows:

$$\hat{r}(\mathbf{x}; \boldsymbol{\theta}_0, \boldsymbol{\theta}_1) \approx \frac{p(\mathbf{x}|\boldsymbol{\theta}_0)}{p(\mathbf{x}|\boldsymbol{\theta}_1)}, \quad (32)$$

where the classifier output $\hat{r}(\mathbf{x}; \boldsymbol{\theta}_0, \boldsymbol{\theta}_1)$ has a specific dependence on the parameter vectors $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$ and the approximation becomes an equality only for a Bayes optimal classifier for each combination. In order to train such classifiers in a data-efficient manner, they suggest using smooth models such as neural networks and a single learning stage based on a large dataset where each observation correspond to an instantiation of the parameters $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_1$ drawn from a reasonable prior distribution $\pi(\boldsymbol{\theta})$ and where \mathbf{x} is drawn from the generative model using those parameters.

Given a flexible enough model and enough training data, the procedure described above could be used to learn a good approximation of the quantity in Eq. (32). For problems where the underlying structure is a mixture model, Cranmer *et al.* also point out that is possible to obtain the quantity $\hat{r}(\mathbf{x}; \boldsymbol{\theta}_0, \boldsymbol{\theta}_1)$ based on the parameterized

output for each pairwise component classification problem which are simpler learning tasks. Because, in practice, the approximation cannot be assumed to be exact, the authors of [17] also propose to have a second stage where generative model samples are used again to calibrate all the relevant values of the parameters as well as a set of diagnostic procedures. They successfully apply this methodology to a set of example problems and discuss its potential usefulness in the context of HEP analysis.

It is worth noting that the component of the vector parameters in θ in CARL could include both nuisance parameters and parameters of interests in the same manner. The nuisance parameters could also be incorporated in the calibration and profiled or marginalized at the inference stage. Once we have a well-calibrated approximation of the likelihood ratio, we can directly use it to construct arbitrary test statistics and confidence intervals for statistical inference. Hence, with the caveats associated with a more involved training procedure and parametric calibration procedure, this technique presents the first principled and general solution for dealing with parameters when using machine learning techniques in the context of HEP inference.

5.3. *Learning more efficiently from the simulator*

As mentioned earlier, one of the caveats of the general applicability of CARL is that the training and probabilistic calibration^h procedure may potentially require a large amount of simulated data to approximate accurately the likelihood ratio $r(\mathbf{x}; \theta_0, \theta_1)$ for all relevant θ_0 and θ_1 when the dimension of θ is large. This practical limitation motivated Brehmer and the original authors of CARL to develop a family of methods [44–46] to estimate the likelihood ratio and other useful quantities for inference in a more data-efficient manner, by augmenting training data with information from the simulator. The

^hThrough this section by calibration we refer to the use of an independent set of simulated data to transform the resulting estimator to ensure its expected statistical properties. We refer to the calibration section of [44] for two different approaches to obtain this type of transformations.

source of the information from the simulations comes from the properties and structure of the data generating process:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) d\mathbf{z} \quad (33)$$

which are characterized by the the joint distribution function $p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})d\mathbf{z}$ where \mathbf{z} are all the latent variables of each observation. In high-energy physics event generation, the joint probability distribution can be factorized in a series of conditional distributions matching the various simulation steps and their dependencies:

$$p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = p(\mathbf{x}|\mathbf{z}_d)p(\mathbf{z}_d|\mathbf{z}_s)p(\mathbf{z}_s|\mathbf{z}_p)\sum_{j=0}^{K-1} p(z_i = j|\boldsymbol{\theta}_{\text{th}})p(\mathbf{z}_p|\boldsymbol{\theta}_{\text{th}}, z_i = j), \quad (34)$$

where $p(z_i = j|\boldsymbol{\theta})$ is the probability of a given type of process j occurring, $p(\mathbf{z}_p|\boldsymbol{\theta}, z_i = j)$ is the conditional probability density of a given set of parton-level four-momenta particles for a given process, $p(\mathbf{z}_s|\mathbf{z}_p)$ is the conditional density of a given parton-shower outcome, $p(\mathbf{z}_d|\mathbf{z}_s)$ is the conditional density of a set of detector interactions and readout noise, and $p(\mathbf{x}|\mathbf{z}_d)$ is the conditional density of a given detector readout. Note that all the factors could depend on additional nuisance parameters; here only the theoretical parameters $\boldsymbol{\theta}_{\text{th}}$ are made explicit for notational simplicity because they are normally the parameters of interest. Also note that the last factor gives rise to the mixture structure mentioned in the last subsection. While $p(\mathbf{x}|\boldsymbol{\theta})$ and ratios of that quantity are typically intractable, the authors suitably remark that the joint likelihood ratio

$$r(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}_0, \boldsymbol{\theta}_1) = \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}_0)}{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}_1)} \quad (35)$$

and the joint score

$$t(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}_0) = \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})|_{\boldsymbol{\theta}_0} \quad (36)$$

can often be obtained exactly for a given simulated observation due to its factorized structure. They propose two regression losses L_r and L_b for each of the previous quantities, which may be used to obtain

an approximation of the likelihood ratio $r(\mathbf{x}|\boldsymbol{\theta}_0, \boldsymbol{\theta}_1)$ and the score $t(\mathbf{x}|\boldsymbol{\theta}_0)$ by empirical risk minimization with various machine learning models such as neural networks. Based on these loss functions, Brehmer *et al.* develop a family of new methods as well as extensions of CARL to more efficiently approximate the parameterized likelihood ratio $r(\mathbf{x}|\boldsymbol{\theta}_0, \boldsymbol{\theta}_1)$ and demonstrate their effectiveness in a few example problems. Another practical innovation developed by the authors, applicable to all the new parameterized likelihood ratio estimators and also to CARL, is that the parameters of the reference hypothesis in $\boldsymbol{\theta}_1$ in Eq. (32) can be kept fixed at an arbitrary value, thus simplifying the learning task significantly. Building upon this work, Stoye and the previous authors [47] also developed two new methods that can incorporate the joint likelihood ratio and the joint score to a loss function based on the cross entropy, which reduces the variance during the learning tasks further improving sample efficiency for obtaining accurate likelihood ratio approximations.

In addition to the efficient techniques for parameterized likelihood ratio estimation discussed above, Brehmer *et al.* [44–46] also propose a new class of methods referred to as SALLY using the regressed score approximation $\hat{t}(\mathbf{x}|\boldsymbol{\theta}_{\text{ref}})$ at a single reference parameter point $\boldsymbol{\theta}_{\text{ref}}$ to construct a summary statistic. The score $t(\mathbf{x}|\boldsymbol{\theta}_{\text{ref}})$, whose dimensionality is the same as that of the parameter vector $\boldsymbol{\theta}$, is a sufficient statistic in the neighborhood of $\boldsymbol{\theta}_{\text{ref}}$, so it is a very useful transformation. Because the dimensionality might still be high in problems with a large number of parameters, they propose to use a one-dimensional projection (i.e. SALLINO) in the direction of parameter variation as alternative lower-dimensional statistic. In the same work, the authors also experiment with augmenting conditional neural density estimators, such as density networks or normalizing flows, with a joint score regression loss function. A calibrated estimation of the likelihood $\hat{p}(\mathbf{x}|\boldsymbol{\theta})$ can be used as a basis for any statistical inference task but its accurate approximation is challenging with a finite data sample, yet many recent advances coming from the field of machine learning in density estimation could eventually make this approach viable.

Similarly to CARL, all these improved techniques for the estimation of likelihood ratios, likelihood scores or the conditional likelihood

itself make no distinction between the statistical parameters in the model. Hence, nuisance parameters can be incorporated in the vector of parameters θ , accounted for like any another parameter in the calibration, and profiled or marginalized at the inference stage. The challenge for their direct application in HEP, particularly for the methods that use augmented data from the simulator, is to approximate or model the effect of all relevant nuisance parameters in the joint likelihood ratio and score. In a recent publication, Brehmer *et al.* [48] presented a software library to simplify the application of these techniques to LHC measurements and included the effect of nuisance parameters from scale and parton distribution function choices by varying the weights associated to each simulated observations.

5.4. Inference-aware summary statistics

With the exception of SALLY (and SALLINO for a fixed projection), the techniques previously discussed are directly based on calibrated likelihood ratios or likelihood approximations, so they are at their core a different form of inference from what is typically done in HEP: they are designed to tackle the inference problem directly, rather than to construct summary statistics. Such a strong paradigm change can be very advantageous but also poses some challenges for its adoption. In recent years, another complementary family of inference-aware techniques has been proposed, whose objective is the construction of machine learning-based summary statistics that are better aligned with the statistical inference goal of HEP analysis, including nuisance parameters. Once constructed, these summary statistics can be used in place of simplified physical summaries or signal vs background classification outputs.

A generic technique in this category, which has direct applicability to HEP analyses, is INFERNO [49]. In that work, authors demonstrate how nonlinear summary statistics can be constructed by minimizing inference-motivated losses via stochastic gradient descent specific for the analysis goal. For example, for an analysis focusing on the measuring of a physical quantity such as a cross-section, the proposed approach can be used to minimize directly, as a loss, an

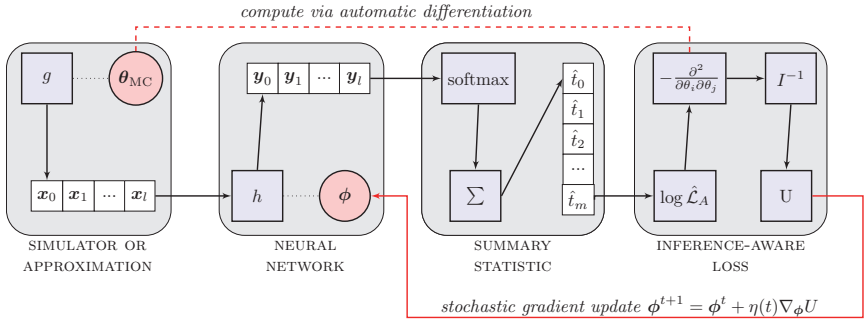


Fig. 5. Learning inference-aware summary statistics. Figure by the authors of INFERNO [49] and licensed under CC BY 4.0.

approximation of the expected uncertainty on the parameter of interest, fully accounting for the effect of relevant nuisance parameters.

In INFERNO and other similar approaches discussed later, the parameters of a neural network are optimized by stochastic gradient descent within an automatic differentiation framework, where the considered loss function accounts for the details of the statistical model as well as the expected effect of nuisance parameters. A graphical depiction of this technique is provided in Fig. 5. The left-most block in the graph refers to sampling a differentiable simulator or approximating the effect of the parameters θ over existing simulated observations, including relevant nuisance parameters. These observations go through a neural network that depends on a set of parameters ϕ (second block from the left) and produces as output a histogram-like summary statistics (third block from the left). Still within the automatic differentiation framework, a synthetic likelihood (e.g., a product of Poisson counts for a histogram-like summary statistic) is constructed. A final inference-aware loss, for example an approximation of the expected uncertainty for the parameters of interest accounting for nuisance parameters, can then be constructed based on the inverse Hessian matrix and used to optimize the neural network parameters.

Note that the approximations used to make a differentiable loss (e.g. continuous relaxation of a histogram) do not affect the rigor of

the resulting statistical inference. Once the summary statistic transformation has been learned with the procedure described above, it can be used, e.g. using an argmax operator instead of a softmax to compute the summary statistic if the approximation of Fig. 5 is used, to carry out statistical inference with the usual procedures and tools as would be done for any other histogram-based summary statistic. The main challenge of using this approach in HEP analyses is that the effect of nuisance parameters has to be included in the auto-differentiation framework, for example by transforming the input features (e.g. momenta and energy calibration uncertainties), by interpolating simulated observation weights (e.g. theoretical and parton distribution function uncertainties) or by considering the interpolation between histogram counts as a last resource. If those challenges can be overcome (even just for part of the nuisance parameters), this method provides an alternative to perform dimensionality reduction using directly an approximation of the inference objective of a given analysis, in contrast with a transformation based on probabilistic classification or a physics-motivated feature. The authors demonstrate the effectiveness of this technique in a multi-dimensional synthetic example with up to three nuisance parameters, where the inference-aware summary statistics outperform even optimal classification-based summaries.

A technique with a similar reach, but that was developed instead for tackling likelihood-free inference problems in astrophysical observations, was presented by Charnock *et al.* [50]. In their work, the authors propose information-maximizing neural networks (IMNN), a machine learning technique to find nonlinear functionals of the data that maximize the Fisher information. The Fisher information during training is computed from the Fisher matrix determinant, that it is itself calculated from the derivatives of the outputs of the network with respect to the parameters of inference at fiducial values by numerical differentiation or directly from the adjoint gradient of a large number of simulations. The authors additionally propose the inclusion of the determinant of the covariance matrix of the neural network outputs in the loss to control the magnitude of the summaries. While they do not consider the problem of nuisance

parameters specifically, their approach will by design find transformations that are minimally affected by nuisance parameters while being maximally sensitive to the parameters. On a related note, Alsing *et al.* [51] develop a useful transformation that can be applied to implicitly marginalize the summary statistics resulting from IMNN or score $t(\mathbf{x}|\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}|\boldsymbol{\theta})$ approximations (e.g. SALLY from the previous subsection).

More recently, there has also been some recent work building upon the ideas behind INFERNO that attempt to simplify its application to high-energy physics analysis or extend its functionality. For example, Wunsch *et al.* [52] suggest using a differentiable transformation of a neural network with a single node to construct a Poisson count likelihood instead of a softmax as the basis for the inference-aware loss. Similarly to what was observed for INFERNO, the authors demonstrate the usefulness of an inference-aware construction in a synthetic example, and also using an extension of the Higgs ML benchmark including nuisance parameters. Following a different path, the authors of NEOS [53] use a technique referred to as fixed-point differentiation to compute gradients of the profile likelihood, thus avoiding the Hessian inverse approximation, and to directly minimize the expected upper limits CL_s . Both Wunsch *et al.* and the authors of NEOS restrict the modeling of the effect of nuisance parameters to histogram interpolation.

In addition to the mentioned approaches, it is worth noting other alternatives with a more limited range of applicability but that could be useful for certain use cases. Elwood *et al.* [54] propose using the expected significance approximation formula for a single bin count experiment, optionally including the effect of a single source of systematic uncertainty directly as a loss of a neural network. For a different type of model, Xia [55] develops a variation of boosted decision tree training referred to as QBDT which targets directly the statistical significance, and which can also include the effect of nuisance parameters in its approximation. In both cases, authors demonstrate with practical examples that the significance optimizing algorithms outperform their classification counterparts.

6. Outlook

The reduction of the effect of systematic uncertainties in parameter estimation is a crucial problem in particle physics. In the past, the problem was attacked by striving for redundancy of the measurement apparatus, robustness of the detection techniques, and the use of analysis methods aiming for inter-calibration, cross-validation, and leveraging as much as possible control datasets and measurements. In the machine learning era, automated methods have become available that may significantly further reduce the impact that imprecise knowledge of latent features of the data have on physics measurements. While already a significant arsenal of techniques has been amassed, no catch-all procedure has emerged yet, so insight is still required to discern the salient features of the problem to be solved and the appropriate method to deploy. The most promising avenues for a general procedure of handling nuisance parameters are those described in Sec. 5, where the optimization objectives are more directly linked to the inference goal.

Acknowledgments

The authors would like to thank Johann Brehmer for some discussions about the techniques included in MADMINER and how they can deal with nuisance parameters. The authors would also like to thank all the anonymous reviewers of this review for their feedback and comments.

References

- [1] P. De Castro Manzano, Statistical learning and inference at particle collider experiments. Ph.D. thesis, University of Padova (2019); <https://cds.cern.ch/record/2701341>.
- [2] ROOT, K. Cranmer, G. Lewis, L. Moneta, A. Shibata and W. Verkerke, HistFactory: A tool for creating statistical models for use with RooFit and RooStats (2012); <https://cds.cern.ch/record/1456844>.
- [3] K. Cranmer, Practical statistics for the LHC, in *2011 European School of High-Energy Physics* (2014); arXiv:1503.07622 [physics.data-an].

- [4] L. Lista, Practical statistics for particle physicists, in *2016 European School of High-Energy Physics* (2017); arXiv:1609.04150 [physics.data-an].
- [5] W. M. Patefield, On the maximized likelihood function, *Sankhyā: Indian J. Statist. Ser. B* **39** (1977) 92.
- [6] D. R. Cox and O. E. Barndorff-Nielsen, *Inference and Asymptotics* (CRC Press, 1994).
- [7] F. James and M. Roos, Minuit — a system for function minimization and analysis of the parameter errors and correlations, *Comput. Phys. Commun.* **10** (1975) 343.
- [8] G. Cowan, K. Cranmer, E. Gross and O. Vitells, Asymptotic formulae for likelihood-based tests of new physics, *Eur. Phys. J. C* **71** (2011) 1554; arXiv:1007.1727 [physics.data-an]. [Erratum: *Eur. Phys. J. C* **73** (2013) 2501].
- [9] J. Thaler and K. Van Tilburg, Maximizing boosted top identification by minimizing N -subjettiness, *J. High Energy Phys.* **02** (2012) 093; arXiv:1108.2701 [hep-ph].
- [10] J. Dolen, P. Harris, S. Marzani, S. Rappoccio and N. Tran, Thinking outside the ROCs: Designing decorrelated taggers (DDT) for jet substructure, *J. High Energy Phys.* **05** (2016) 156; arXiv:1603.00027 [hep-ph].
- [11] I. Moul, B. Nachman and D. Neill, Convolved substructure: Analytically decorrelating jet substructure observables, *J. High Energy Phys.* **05** (2018) 002; arXiv:1710.06859 [hep-ph].
- [12] R. M. Neal, Computing likelihood functions for high-energy physics experiments when distributions are defined by simulators with nuisance parameters, <http://cds.cern.ch/record/1099977>.
- [13] CDF, D0, T. Aaltonen *et al.*, Evidence for a particle produced in association with weak bosons and decaying to a bottom-antibottom quark pair in Higgs boson searches at the Tevatron, *Phys. Rev. Lett.* **109** (2012) 071804; arXiv:1207.6436 [hep-ex].
- [14] CMS, S. Chatrchyan *et al.*, Combined results of searches for the standard model Higgs boson in pp collisions at $\sqrt{s} = 7$ TeV, *Phys. Rev. Lett. B* **710** (2012) 26; arXiv:1202.1488 [hep-ex].
- [15] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski and D. Whiteson, Parameterized neural networks for high-energy physics, *Eur. Phys. J. C* **76** (2016) 235; arXiv:1601.07913 [hep-ex].
- [16] DELPHES 3, J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens and M. Selvaggi, DELPHES 3, A modular framework for fast simulation of a generic collider experiment, *J. High Energy Phys.* **02** (2014) 057; arXiv:1307.6346 [hep-ex].
- [17] K. Cranmer, J. Pavez and G. Louppe, Approximating likelihood ratios with calibrated discriminative classifiers (2015); arXiv:1506.02169 [stat.AP].
- [18] J. Aguilar-Saavedra, J. H. Collins and R. K. Mishra, A generic anti-QCD jet tagger, *J. High Energy Phys.* **11** (2017) 163; arXiv:1709.01087 [hep-ph].
- [19] S. Chang, T. Cohen and B. Ostdiek, What is the machine learning?, *Phys. Rev. D* **97** (2018) 056009; arXiv:1709.10106 [hep-ph].

- [20] K. Datta and A. Larkoski, How much information is in a jet?, *J. High Energy Phys.* **06** (2017) 073; arXiv:1704.08249 [hep-ph].
- [21] R. Dalitz, On the analysis of tau-meson data and the nature of the tau-meson, *Phil. Mag. Ser.* **44** (1953) 1068.
- [22] J. Stevens and M. Williams, uBoost: A boosting method for producing uniform selection efficiencies from multivariate classifiers, *J. Instrum.* **8** (2013) P12013; arXiv:1305.7248 [nucl-ex].
- [23] Y. Freund and R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, *J. Comput. Syst. Sci.* **55** (1997) 119.
- [24] A. Rogozhnikov, A. Bukva, V. Gligorov, A. Ustyuzhanin and M. Williams, New approaches for boosting to uniformity, *J. Instrum.* **10** (2015) T03002; arXiv:1410.4140 [hep-ex].
- [25] G. Kasieczka and D. Shih, DisCo fever: Robust networks through distance correlation (2020); arXiv:2001.05310 [hep-ph].
- [26] S. Wunsch, S. Jörger, R. Wolf and G. Quast, Reducing the dependence of the neural network function to systematic uncertainties in the input space (2019); arXiv:1907.11674 [physics.data-an].
- [27] C. Adam-Bourdarios, G. Cowan, C. Germain-Renaud, I. Guyon, B. Kégl and D. Rousseau, The Higgs machine learning challenge, *J. Phys. Conf. Ser.* **664** (2015) 072015.
- [28] S. Ben-David, J. Blitzer, K. Crammer and F. Pereira, Analysis of representations for domain adaptation, in *Advances in Neural Information Processing Systems 19*, eds. B. Schölkopf, J. C. Platt, and T. Hoffman (MIT Press, 2007), pp. 137–144.
- [29] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira and J. W. Vaughan, A theory of learning from different domains, *Machine Learn.* **79** (2009) 151.
- [30] H. Ajakan, P. Germain, H. Larochelle, F. Laviolette and M. Marchand, Domain-adversarial neural networks (2014); arXiv:1412.4446.
- [31] G. Louppe, M. Kagan and K. Cranmer, Learning to pivot with adversarial networks, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017), pp. 981–990.
- [32] M. H. DeGroot and M. J. Schervish, *Probability and Statistics* (Pearson Education, 2012).
- [33] C. Shimmin, P. Sadowski, P. Baldi, E. Weik, D. Whiteson, E. Goul and A. Søgaard, Decorrelated jet substructure tagging using adversarial neural networks, *Phys. Rev. D* **96** (2017) 074034; arXiv:1703.03507 [hep-ex].
- [34] V. Estrade, C. Germain, I. Guyon and D. Rousseau, Adversarial learning to eliminate systematic errors: A case study in high energy physics, in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017).
- [35] P. Simard, B. Victorri, Y. LeCun and J. Denker, Tangent prop — a formalism for specifying selected invariances in an adaptive network, in *Advances in*

- Neural Information Processing Systems 4*, eds. J. E. Moody, S. J. Hanson, and R. P. Lippmann, (Morgan-Kaufmann, 1992), pp. 895–903.
- [36] A. Blance, M. Spannowsky and P. Waite, Adversarially-trained autoencoders for robust unsupervised new physics searches, *J. High Energy Phys.* **10** (2019) 047; arXiv:1905.10384 [hep-ph].
 - [37] C. Englert, P. Galler, P. Harris and M. Spannowsky, Machine learning uncertainties with adversarial neural networks, *Eur. Phys. J. C* **79** (2019) 4; arXiv:1807.08763 [hep-ph].
 - [38] L. M. Dery, B. Nachman, F. Rubbo and A. Schwartzman, Weakly supervised classification in high energy physics, *J. High Energy Phys.* **2017** (2017) 145.
 - [39] E. M. Metodiev, B. Nachman and J. Thaler, Classification without labels: Learning from mixed samples in high energy physics, *J. High Energy Phys.* **2017** (2017) 174.
 - [40] T. Cohen, M. Freytsis and B. Ostdiek, (machine) learning to do more with less, *J. High Energy Phys.* **2018** (2018) 34.
 - [41] P. T. Komiske, E. M. Metodiev, B. Nachman and M. D. Schwartz, Learning to classify from impure samples with high-dimensional data, *Phys. Rev. D* **98** (2018) 011502.
 - [42] K. Cranmer, J. Brehmer and G. Louppe, The frontier of simulation-based inference (2019); arXiv:1911.01429 [stat.ML].
 - [43] J. Neyman and E. S. Pearson, On the problem of the most efficient tests of statistical hypotheses, *Philos. Trans. Roy. Soc. London Ser. A* **231** (1933) 289.
 - [44] J. Brehmer, K. Cranmer, G. Louppe and J. Pavez, A guide to constraining effective field theories with machine learning (2018); arXiv:1805.00020 [hep-ph].
 - [45] J. Brehmer, G. Louppe, J. Pavez and K. Cranmer, Mining gold from implicit models to improve likelihood-free inference (2018); arXiv:1805.12244 [stat.ML].
 - [46] J. Brehmer, K. Cranmer, G. Louppe and J. Pavez, Constraining effective field theories with machine learning (2018); arXiv:1805.00013 [hep-ph].
 - [47] M. Stoye, J. Brehmer, G. Louppe, J. Pavez and K. Cranmer, Likelihood-free inference with an improved cross-entropy estimator (2018); arXiv:1808.00973 [stat.ML].
 - [48] J. Brehmer, F. Kling, I. Espejo and K. Cranmer, MadMiner: Machine learning-based inference for particle physics, *Comput. Softw. Big Sci.* **4** (2020) 3; arXiv:1907.10621 [hep-ph].
 - [49] P. de Castro and T. Dorigo, INFERNO: Inference-aware neural optimisation, *Comput. Phys. Commun.* **244** (2019) 170.
 - [50] T. Charnock, G. Lavaux and B. D. Wandelt, Automatic physical inference with information maximizing neural networks, *Phys. Rev. D* **97** (2018) 083004.
 - [51] J. Alsing and B. Wandelt, Nuisance hardened data compression for fast likelihood-free inference, *Mon. Not. R. Astron. Soc.* **488** (2019) 5093.

- [52] S. Wunsch, S. Jörger, R. Wolf and G. Quast, Optimal statistical inference in the presence of systematic uncertainties using neural network optimization based on binned poisson likelihoods with nuisance parameters (2020); arXiv:2003.07186 [physics.data-an].
- [53] L. Heinrich and N. Simpson, pyhf/neos: Initial zenodo release (2020); doi: 10.5281/zenodo.3697981.
- [54] A. Elwood and D. Krücker, Direct optimisation of the discovery significance when training neural networks to search for new physics in particle colliders (2018); arXiv:1806.00322 [hep-ex].
- [55] L.-G. Xia, QBDT, a new boosting decision tree method with systematic uncertainties into training for high energy physics (2018); arXiv:1810.08387 [physics.data-an].

This page intentionally left blank

Chapter 18

Bayesian Neural Networks

Tom Charnock^{*,†,**}, Laurence Perreault-Levasseur^{‡,§,¶,††}
and François Lanusse^{||,‡‡}

^{*}*Sorbonne Université, CNRS, UMR 7095,
Institut d'Astrophysique de Paris*

[†]*98 bis bd Arago, 75014 Paris, France*

[‡]*Department of Physics, Université de Montréal,
Montréal, Canada*

[§]*Mila-Quebec Artificial Intelligence Institute,
Montréal, Canada,*

[¶]*Center for Computational Astrophysics,
Flatiron Institute, New York, USA*

^{||}*AIM, CEA, CNRS, Université Paris-Saclay,
Université Paris Diderot, Sorbonne Paris Cité,
F-91191 Gif-sur-Yvette, France*

^{**}*tom@charnock.fr*

^{††}*llevasseur@astro.umontreal.ca*

^{‡‡}*francois.lanusse@cea.fr*

In recent times, neural networks have become a powerful tool for the analysis of complex and abstract data models. However, their introduction intrinsically increases our uncertainty about which features of the analysis are model-related and which are due to the neural network. This means that predictions by neural networks have biases which cannot be trivially distinguished from being due to the true nature of the creation and observation of data or not. In order to attempt to address such issues we discuss Bayesian neural networks: neural networks where the uncertainty due to the network can be characterized. In particular, we outline the Bayesian statistical framework which allows us to categorize uncertainty in terms of the ingrained randomness of observing certain data and the uncertainty from our *lack of knowledge* about how processes that are observed can occur. In presenting such techniques, we show how uncertainties which arise in the predictions made by neural networks can be characterized in principle. We provide descriptions of

the two favored methods for analyzing such uncertainties. We will also describe how both of these methods have substantial pitfalls when put into practice, highlighting the need for other statistical techniques to truly be able to do inference when using neural networks.

1. Introduction

In recent times, we have seen the power and ability that neural networks and deep learning methods can provide for fitting abstractly complex data models. However, any prediction from a neural network is necessarily and unknowably biased due to factors such as: choices in network architecture; methods for fitting networks; cuts in sets of training data; uncertainty in the distribution of realistic data; and lack of knowledge about the physical processes which generate such data. In this chapter, we elucidate ways in which one can learn how to separate, as much as possible, the sources of error which are due to intrinsic distribution of observed data and those that we have introduced by modeling this distribution both with physical models and by considering neural networks as statistical models.

1.1. *The need for statistical modeling*

Imagine that we walk into a room and there are ten, six-sided dice whose result we observe. The dice are then taken away and we are left to wonder how likely is it that we observed that particular roll. Because the dice have been taken away we cannot perform any repeated experiments to make simple estimates of the probability of what we assume is a random process based on counts. Instead, we can build a model describing the dice roll and infer the values of the parameters of this model based on how much evidence can be obtained from the observation. We could assume that all the dice were equally weighted and there were no external factors to affect the roll and therefore suggest that the result of the dice roll follow a multinomial distribution with equal probability for each of the six possible results from each of the ten dice. However, what if we had observed nine dice showing one and the other die showing six? It would be very unlikely to observe such an event within this model, in fact we can calculate the probability of this result in this model

to be 0.000017%. We could instead decide that each dice is weighted so that there is a 90% chance that they will land on one and a 10% chance that they will land on six, in which case the probability of observing this event is much higher at $\sim 38\%$. Or, we could decide that nine of the dice are weighted so that there is a 100% chance that they will land on one and the tenth die has a 100% chance that it will land on six, and in which case the observed event is certain. The problem is that we do not know about the state of the dice or the processes by which different results can be obtained. Therefore, we do not know the values of the parameters in the multinomial model that we use to describe how likely any result is and so there is a source of uncertainty in any prediction we make.

1.2. *Aleatoric and epistemic uncertainty*

Uncertainty can be categorized into two classes: aleatoric and epistemic. These two uncertainties explain, respectively, scatter from what we cannot know and error due to lack of knowledge. For example, we do not know how likely it is to have observed nine ones and one six on ten six-sided dice when we do not have access to those dice. This is an intrinsic uncertainty due to the random nature of the way the observed event happens and the way we make observations. As such, we call this uncertainty *aleatoric* since it cannot be reduced through greater understanding. On the other hand, when we are trying to understand a particular set of observations there are things we do not know but could, in principle, learn about: what are the properties of a physical processes which are necessary to create such data? what types of distribution could describe how likely were we to see such an observation? and how certain are we that such a model is supported by our data? For the dice roll example, we do not know if the dice are weighted, or if weighted dice would better fit the observed result, or if there is something that we are not considering, like whether the dice were placed in a particular way rather than thrown and being the result of a random process. By addressing the above questions, we can narrow down on the possible ways to describe the observation and learn about the state of how the observation came to be through the use of the available data.

For example, knowledge about the result observed on the rolled dice can allow us to narrow down the possible values of the probabilities in the multinomial model that could produce such an observation, therefore reducing our uncertainty. We call this reducible uncertainty *epistemic*.

Whilst a simple example, such as the rolling of dice, seems trivial, it describes any way of learning about our surroundings using the available data. Every experiment performed exists in a single universe that has undergone epochs of evolution and its constituent particles and forces have interacted to provide us with what we can observe. There is therefore aleatoric uncertainty due to the fact that we can only observe this one realization of our universe, and we cannot observe other universes to increase our knowledge about how likely our universe is to be the way it is. We can, though, make models which describe the constituents of the universe, the way they interact and the evolution to get what we see today. Although we do not know how likely the observed data is, we can reduce our uncertainty about the possible models, and its parameters values, which are supported by the data. In fact, even repetitions within a single experiment are taking place at different locations and times in the same single universe — therefore, it is only an assumption of the model for analyzing the repeated experiment that any observations are independent results and is not intrinsic to the data that we observe.

The use of a neural network for the analysis of data modifies our data model to include any effects that are introduced by the network. There is, therefore, intrinsic (aleatoric) uncertainty due to the stochastic nature of the data, and epistemic uncertainty now due to both the lack of knowledge about process generating the data as well as the design of the neural network, the way it is trained, the choice of cost function, etc. Neural networks should therefore be seen as an extended, extra-parameterized physical model for the data, whose parameters can be inferred through the support of data. This means, to be able to use networks to make scientifically relevant predictions, the epistemic uncertainty must be well understood and properly characterized.

For the most part, estimates of how well a neural network generalizes are obtained using large sets of validation and testing data. It is then common to suggest a neural network “works” when there is a strong, but handwavy, relative agreement. However, these neural networks do not address the probability that any prediction coincides with the truth. There is no separation between aleatoric and epistemic uncertainty and no knowledge of how likely (or well) a new example of data is to provide a realistic prediction. It is *possible*, though, to quantify this epistemic error caused by our lack of knowledge about the properties of a neural network, and characterizing this uncertainty can allow us to perform reasoned inference. In this chapter, we will lay down the formalism for Bayesian neural networks: treating neural networks as statistical models whose parameters are attributed probabilities as a degree of belief which can be logically updated under the support from data. In such a form, neural networks can be used to make statements of inference about how likely we are to believe the outputs of neural networks, reducing the lack of trust that is inherent in the standard deep learning setup. We will also show some ways of practically implementing this Bayesian formalism with examples from astronomy and cosmology.

2. Bayesian Neural Networks

In this section, we will show how one can use a Bayesian statistical framework to assess both aleatoric and epistemic uncertainty in a model which includes neural networks, and describe how epistemic uncertainty can be reduced under the evidence of supporting data using Bayesian inference.

2.1. Bayesian statistics

When speaking of *uncertainty*, we are really describing our lack of knowledge about the truth. This uncertainty is subjective, in that it is not an inherent property of a problem but rather the way we construct the problem. If we are uncertain about the results of a particular experiment, we do not know exactly what the result of that

experiment will be. The Bayesian (or subjective) statistical framework is a scientific viewpoint in which we admit that we do not (and are not able to) know the truth about any particular hypothesis. Our uncertainty, or our degree of belief in the truth, is attributed probabilities, i.e. hypotheses we believe more strongly are described as being more likely. Of course, in this construction, probabilities can vary from person to person, since different beliefs can be held by different people. Without any *prior* knowledge, we are free to believe what we will. However, by using Bayesian inference, we are able to reduce epistemic uncertainty and update our *a priori* knowledge by obtaining evidence, *a posteriori*. It is important to realize that, whilst our *a priori* beliefs describe the epistemic uncertainty, this quantification can be artificially small without the support of observations. If our beliefs are not supported by the evidence, then the *a posteriori* probability describing the state of our belief after obtaining evidence will become more uncertain, which is a better characterization of the state of our knowledge. Under repeated application of new evidence, we can update our beliefs to hone in on the best supported result.

2.1.1. *Statistical models*

A Bayesian statistical framework is a natural setting to build models with which we can infer the most likely distributions and underlying processes that generate some observable events. Observations can be thought of as existing in measurable space of possible events, $(\mathcal{S}, \mathcal{E}, \mathcal{P})$. The first element is the sampling space, \mathcal{S} , which describes the set of all possible outcomes for a given problem.^a Each outcome is a random variable, $d \in \mathcal{S}$, whose value is a single measured observation or result. An event, $\mathcal{D} \subset \mathcal{S}$, is defined as a subset of all possible outcomes. The set of all events that can possibly occur is \mathcal{E} . Referring back to Sec. 1.1, we can think of the sampling space, \mathcal{S} , as the set of any possible roll of a six-side dice and the value of any roll as an outcome denoted by the value of $d \in \mathcal{S}$. An event, \mathcal{D} , could then be a collection of different outcomes, such as the nine ones and

^aFor a more in depth discussion of the measure theoretic definition of probability, see works such as [1] or other graduate level texts.

one six observed on the ten dice described in Sec. 1.1. A statistical description of data also has a measure on the space of possible events, $\mathcal{P} : \mathcal{D} \in \mathcal{E} \mapsto \mathcal{P}(\mathcal{D}) \in [0, 1]$, which is a function that assigns a value between 0 and 1 to every event, $\mathcal{D} \in \mathcal{E}$, describing how likely it is for such an event to occur. This probability indicates that an event, \mathcal{D} , is impossible when $\mathcal{P}(\mathcal{D}) = 0$ and is certain when $\mathcal{P}(\mathcal{D}) = 1$. The measure, \mathcal{P} , of this measurable space is additive, so that it is certain that any *possible* event *can* occur, $\mathcal{P}(\mathcal{E}) = 1$.

While we can observe some subset of all possible outcomes from this probability space we do not necessarily know *which* particular outcomes we will observe from the random processes generating any \mathcal{D} . That is, given the value of some observed event \mathcal{D} , sampled from \mathcal{E} with a probability \mathcal{P} , i.e. from the measurable space $(\mathcal{S}, \mathcal{E}, \mathcal{P})$, we would not know which event would occur from the distribution, \mathcal{P} . Even if we knew exactly about the dice, how they were weighted, how hard they were thrown, etc. we would still not know exactly which result we would observe if the process had some random aspect. The uncertainty due to the statistical nature of the data generation cannot be reduced or learned about and is therefore *aleatoric uncertainty*.

It is the endeavor of science to find models which allow us to describe the things we observe and therefore be able to make predictions using these models. In practice, we cannot know the form of \mathcal{P} and as such we attempt to model the probability measure using a statistical model $(\mathcal{S}_\alpha, \mathcal{E}_\alpha, \mathcal{P})$. In a Bayesian context, \mathcal{S}_α is another sampling space of possible parameterized distributions with an outcome, $\alpha \in \mathcal{S}_\alpha$, representing all properties of a particular distribution, i.e. functional form, shape, as well as the possible values of some unobservable random variables, $\omega \in \Omega_\alpha$, which generate $d \in \mathcal{S}$, etc. Any possible set of $\alpha \in \mathcal{S}_\alpha$ is an event in the space of possible distributions, $\mathcal{a} \in \mathcal{E}_\alpha$, which can model $(\mathcal{S}, \mathcal{E}, \mathcal{P})$. Effectively, any $\mathcal{a} \in \mathcal{E}_\alpha$ defines a model $((\mathcal{S}, \Omega_{\mathcal{a}}), (\mathcal{E}, \mathcal{E}_\omega), \mathcal{P}_{\mathcal{a}})$ of $(\mathcal{S}, \mathcal{E}, \mathcal{P})$. That is, any $\mathcal{a} \in \mathcal{E}_\alpha$ introduces a sampling space of unobservable random variables, $\Omega_{\mathcal{a}}$, whose values, $\omega \in \Omega_{\mathcal{a}}$, can generate outcomes, $d \in \mathcal{S}$. \mathcal{E}_ω then defines the set of all possible unobservable random variables, $\omega \subset \Omega_\alpha$, which can generate events, $\mathcal{D} \in \mathcal{E}$.

Considering the dice rolling problem, $\alpha \in \mathcal{E}_\alpha$ could be the use of a multinomial to model the probability of the distribution of possible results, \mathcal{D} , from throwing 10 dice. In this case, one choice of α could be that, say, there are six model parameters per die, $w \in \mathcal{E}_w$, which ascribe the probability that each side of each die would land face up. Any $\alpha \in \mathcal{E}_\alpha$ also defines a probability measure, $p_\alpha : (\mathcal{D}, w) \in (\mathcal{E}, \mathcal{E}_w) \mapsto p_\alpha(\mathcal{D}, w) \in [0, 1]$, describing how likely any observable-event-and-unobservable-parameter pairs are, i.e. how likely any value of the parameters, w , is to give rise to some observation, \mathcal{D} , is described by the value of the joint distribution of observables and parameters, $p_\alpha(\mathcal{D}, w)$. The possible parameterized statistical model characterizes what physical processes generate an observable outcome, our assumption about the possible values of the parameters of those physical processes, and how likely we are to obtain any set of outcomes *and* physical parameters.

We assign a probabilistic degree of assumption about the possible models from our *prior* knowledge, $p : \alpha \in \mathcal{E}_\alpha \mapsto p(\alpha) \in [0, 1]$, that any set of possible distributions, $\alpha \in \mathcal{E}_\alpha$, encapsulates the underlying probability measure, \mathcal{P} , describing the probability of events, \mathcal{D} , occurring. For example, we might believe, thanks to our prior knowledge of the problem, that a model, $\alpha^* \in \mathcal{E}_\alpha$, describing the probability of outcomes of dice roll as a multinomial distribution with equal parameter values is more likely to be correct than another model, $\alpha^\dagger \in \mathcal{E}_\alpha$, which uses, say, a Dirichlet distribution. In this case, we would ascribe the probability of α^* as being more likely than α^\dagger , i.e. $p(\alpha^*) > p(\alpha^\dagger)$. The lack of knowledge about the possible values of α is the source of *epistemic* uncertainty. Whilst we will never know the exact distribution of data from $(\mathcal{S}, \mathcal{E}, \mathcal{P})$, we can increase our knowledge about how to model it with $\alpha \in \mathcal{E}_\alpha$ under the evidence of observed events, $\mathcal{D} \in \mathcal{E}$, thereby reducing the epistemic uncertainty.

Since the unobservable parameters, $w \in \mathcal{E}_w$, generate possible sets of observable outcomes, $\mathcal{D} \in \mathcal{E}$, we can write down how likely we are to observe some event, \mathcal{D} , *given* that the unobservable parameters, w , have a particular value,

$$p_\alpha(\mathcal{D}, w) = \mathcal{L}(\mathcal{D}|w)p_\alpha(w). \quad (1)$$

We call $\mathcal{L} : (\mathcal{D}, w) \in (\mathcal{E}, \mathcal{E}_w) \mapsto \mathcal{L}(\mathcal{D}|w) \in [0, 1]$ the *likelihood* of some values of observables \mathcal{D} , given the values of parameters, w , and $p_a : w \in \mathcal{E}_w \mapsto (w) \in [0, 1]$ is the *a priori* (or prior) distribution of parameters describing what we assume the values of w to be based on our current knowledge. Therefore, some (but not all) of the epistemic uncertainty is encapsulated by p_a . The prior distribution, p_a , does not, however, describe the form of the parameterized joint distribution, p_a , modeling, $(\mathcal{S}, \mathcal{E}, \mathcal{P})$, and so we must also consider how likely is it that we assume our choice of possible distributions, $p(\mathcal{a})$, to properly characterize the epistemic uncertainty.

2.1.2. Bayesian inference

By observing events, $\mathcal{D} \in \mathcal{E}$, we can update our assumptions about the values of any set of unobservable random variables, $w \in \mathcal{E}_w$, and distributions, $\mathcal{a} \in \mathcal{E}_a$, correctly modeling the probabilistic space, $(\mathcal{S}, \mathcal{E}, p)$, for some problem. This is how we can reduce our epistemic uncertainty. The probability describing our choice of assumptions in the possible values of w and \mathcal{a} obtained *after* we have observed an event, \mathcal{D} , is called the *a posteriori* (or posterior) distribution, $\rho : (\mathcal{D}, w) \in (\mathcal{E}, \mathcal{E}_w) \mapsto \rho(w|\mathcal{D}) \in [0, 1]$ and can be derived by expanding the joint distribution

$$\begin{aligned} p_a(\mathcal{D}, w)p(\mathcal{a}) &= \mathcal{L}(\mathcal{D}|w)p_a(w)p(\mathcal{a}) \\ &= \rho(w|\mathcal{D})e(\mathcal{D})p(\mathcal{a}), \end{aligned} \quad (2)$$

and equating both sides to get Bayes' theorem

$$\rho(w|\mathcal{D}) = \frac{\mathcal{L}(\mathcal{D}|w)p_a(w)}{e(\mathcal{D})}. \quad (3)$$

This equation tells us that, given a particular parameterized model, \mathcal{a} , the probability that some parameters, w , have a particular value when some event, \mathcal{D} , is observed is proportional to the likelihood of the observation of such an event given a particular value of the parameters, w , generating the event. The probability of those parameter values is described by our belief in their value, $p_a(w)$. The *evidence*, $e : \mathcal{D} \in \mathcal{E} \mapsto e(\mathcal{D}) \in [0, 1]$, that the parameterized

distribution accurately describes the distribution of some event is

$$e(\mathcal{D}) = \int_{\mathcal{E}_\omega} d\omega \mathcal{L}(\mathcal{D}|\omega) p_a(\omega). \quad (4)$$

If the probability of \mathcal{D} is small when the likelihood is integrated over all possible sets of parameter values, $\omega \in \mathcal{E}_\omega$, both of which are defined by a , then there is little support for that choice of a value of $a \in \mathcal{E}_a$. This would suggest that we need to update our assumptions about the parameterized distribution, $p(a)$, being able to represent the true model, $(\mathcal{S}, \mathcal{E}, \mathcal{P})$.

Maximum likelihood estimation: In classical statistics, the unobserved random variables, $\omega \in \mathcal{E}_\omega$, are considered to be fixed parameters of a particular statistical model, $a \in \mathcal{E}_a$. The parameters which best describes some event, \mathcal{D} , can be found maximizing the likelihood function

$$\hat{\omega} = \arg \max_{\omega \in \mathcal{E}_\omega} \mathcal{L}(\mathcal{D}|\omega). \quad (5)$$

Although this point in parameter space maximizes the likelihood and can be found fairly easily by various optimization schemes, it is completely ignorant about the shape of the distribution, $\mathcal{L}(\mathcal{D}|\omega)$. Moreover, we do not even consider quantifying how likely we think any particular value of ω (and a) are. This means that the possible parameters values are degenerated to one point and absolute certainty is ascribed to a choice of model and its parameters. Furthermore, for skewed distributions, the mode of the likelihood can be far away from the expectation value (or mean) of the distribution and therefore the maximum likelihood estimate might not even be representative. Any epistemic uncertainty in the model is ignored since we do not consider our degree of belief in the model parameters, ω , once they are fixed at the maximum of the likelihood, nor do we quantify how confident we are about the particular statistical model a considered.

Maximum *a posteriori* estimation: The simplest form of Bayesian inference is finding the maximum *a posteriori* (MAP) estimate, i.e. the mode of the posterior distribution for a given

model, α , as

$$\begin{aligned}\hat{w} &= \arg \max_{w \in \mathcal{E}_\alpha} \rho(w|\mathcal{D}) \\ &= \arg \max_{w \in \mathcal{E}_\alpha} \mathcal{L}(\mathcal{D}|w)p_\alpha(w).\end{aligned}\tag{6}$$

Note that, when we think that any values of the model parameters are equally likely, i.e. the prior distribution, $p_\alpha(w)$, is uniform, then $\mathcal{L}(\mathcal{D}|w) \propto \rho(w|\mathcal{D})$ and MAP estimation is equivalent to maximum likelihood estimation. So, whilst MAP estimation is Bayesian due to the addition of our belief in possible parameter values, $p_\alpha(w)$, this form of inference suffers in exactly the same way that maximum likelihood estimation does: the mode of the posterior might also be far from the expectation value and not be representative, and all information about the epistemic uncertainty is underestimated because knowledge about the distribution of parameters is ignored.

Bayesian posterior inference: To effectively characterize the epistemic uncertainty, not only should we consider Bayes' theorem (3), one should work with the marginal distribution over the prior probability of parameterized models

$$\begin{aligned}e(\mathcal{D}) &= \int_{\mathcal{E}_\alpha} d\alpha \, e(\mathcal{D})p(\alpha), \\ &= \int_{\mathcal{E}_\alpha} \int_{\mathcal{E}_w} d\alpha dw \, \mathcal{L}(\mathcal{D}|w)p_\alpha(w)p(\alpha).\end{aligned}\tag{7}$$

Practically, the space of possible models, \mathcal{E}_α , can be infinitely large, although our belief in possible models, $p(\alpha)$, does not have to be. Still, the integration over all possible models often makes the calculation of $e(\mathcal{D})$ effectively intractable. In practice, we tend to choose a particular model and, in the best case (where we have lots of time and computational power) use empirical Bayes to calculate the mode of the possible marginal distributions

$$\begin{aligned}\hat{\alpha} &= \arg \max_{\alpha \in \mathcal{E}_\alpha} e(\mathcal{D})p(\alpha), \\ &= \arg \max_{\alpha \in \mathcal{E}_\alpha} \int_{\mathcal{E}_w} dw \, p_\alpha(\mathcal{D}|w)p_\alpha(w)p(\alpha).\end{aligned}\tag{8}$$

As with the MAP estimate of the parameters, \hat{a} describes the most likely believed model that supports an event, \mathcal{D} . However, again as with the MAP estimate of the parameters, a model, $a = \hat{a}$, might have artificially small epistemic uncertainty due to discarding the rest of the knowledge of the distribution. To be able to correctly estimate this epistemic uncertainty, one must update, logically, the probability of any possible models and parameters based on the acquisition of knowledge.

2.2. *Neural networks formulated as statistical models*

We can consider neural networks as part of a statistical model. In this case, we usually think of an observable outcome as a pair of input and target random variable pairs,^b $d = (x, y) \in \mathcal{S}$. An event is then a subset of pairs $\mathcal{D} = (x, y) \in \mathcal{E}$ with probability $\mathcal{P}(x, y)$. We can then use a neural network as a parameterized, nonlinear function

$$r = f_{w,a}(x) \quad (9)$$

where r are considered the parameters of a distribution which models the likelihood of targets given inputs, $\ell(y|x, w) = \mathcal{L}(x, y|w)/e(x)$. The form of the function, i.e. the architecture, the number, value and distribution of network parameters $w \in \mathcal{E}_w$, initialization of the network, etc. is described by some hyperparameters, $a \in \mathcal{E}_a$. The prescription for this likelihood, $\ell(y|x, w)$, can range from being defined as $\ell(y|x, w) \propto \exp[-\Lambda(y, r)]$, where $\Lambda(y, r)$ is an unregularized loss function measuring the similarity of the output of a neural network, r , to some target,^c y , to parametric distributions such as a mixture of distributions or neural density estimators.

^bAlthough we discuss pairs x and y suggesting *inputs* and *targets*, note that this notation is generic. For example, for auto-encoders, we would consider the target to be equivalent to the input, and for generative networks we would consider the input to be some latent variables with which to generate some targets, etc.

^cFor example, a classical mean squared loss corresponds to modeling the negative logarithm of the likelihood as a simple standard unit variance diagonal (multivariate) Gaussian with a mean at the neural network output, r .

When considering a neural network as an abstract function, it can be possible to obtain virtually any value of r for a given input x at any values of the network parameters, w , since the network parameters are often unidentifiable [2] and the functional form of the possible values of r is very likely infinite in extent and no statement about convexity can be made. The reason why we use neural networks is because we can carve out parts of useful parameter space which provide the function which describes how to best fit some known data, (x, y) , using the likelihood, $\ell(y|x, w)$, as defined by the data itself. We normally describe this set of known data which ascribes acceptable regions of parameter space where the likelihood makes sense as a *training* set, $(x, y)_{\text{train}} \in \mathcal{E}$. However, evaluating the neural network to get $r = f_{w,a}(x)$ and assuming that the output, r , has sensible values to correctly define the form of the likelihood of the sampling distribution of targets will often be misleading.^d This statement is true for *any* value of the network parameters, $w \in \mathcal{E}_w$, since most values of w do not correspond to neural networks which perform the desired function.

Having described neural networks as statistical models we can, further, place them in a Bayesian context by associating a probabilistic quantification of our assumptions, $p_a(w)$, to the values of the network parameters, $w \in \mathcal{E}_w$, for a network $a \in \mathcal{E}_a$, which we believe to be able to represent the true distribution of observed events, $\mathcal{P}(x, y)$, with probability $p(a)$. $p(a)$ (and the associated $p_a(w)$) represent the epistemic uncertainty due to the neural network, whilst the aleatoric uncertainty arises due to the fact that it is not known exactly which (x, y) would arise from the statistical model $(\mathcal{S}, \mathcal{E}, \mathcal{P})$. We can use Bayesian statistics to update our beliefs and obtain posterior predictive estimates of targets, y , based on this

^dA sensible likelihood for network targets can be created by making the parameters of the network identifiable. One such method is to use *neural physical engines* [3], where neural networks are designed using physical motivation for the parameters. However, there is a trade-off with this identifiability which comes at the expense of fitting far less complex functions than are usually considered when using neural networks, but far less data and energy is needed to train such models.

information via the posterior predictive distribution

$$p(\mathbf{y}|\mathbf{x}) = \int_{\mathcal{E}_\alpha} \int_{\mathcal{E}_w} d\alpha dw \ell(\mathbf{y}|\mathbf{x}, w) p_\alpha(w) p(\alpha). \quad (10)$$

By integrating over all possible parameters for all possible network choices, we obtain a distribution describing how probable different values of \mathbf{y} are, from our model, which incorporates our lack of knowledge.

The region where we assume that the parameters allow the network to perform its intended purpose is described by, $\rho(w|(\mathbf{x}, \mathbf{y})_{\text{train}})$. This is our first step in the Bayesian inference. Bayes' theorem tells us

$$\rho(w|(\mathbf{x}, \mathbf{y})_{\text{train}}) = \frac{\ell(\mathbf{y}_{\text{train}}|\mathbf{x}_{\text{train}}, w) p_\alpha(w)}{e((\mathbf{x}, \mathbf{y})_{\text{train}})}, \quad (11)$$

so that updating our knowledge of the parameters given the presence of a training set allows us to better characterize the probability of obtaining \mathbf{y} from \mathbf{x} with a particular neural network

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}, (\mathbf{x}, \mathbf{y})_{\text{train}}) &= \int_{\mathcal{E}_\alpha} \int_{\mathcal{E}_w} d\alpha dw \ell(\mathbf{y}|\mathbf{x}, w) \\ &\times \rho(w|(\mathbf{x}, \mathbf{y})_{\text{train}}) p(\alpha). \end{aligned} \quad (12)$$

To encapsulate the uncertainty in the network, we need to calculate the posterior distribution of network parameters, w , as in (11), which we can then use to calculate the distribution of possible \mathbf{y} as described by the predicted \mathbf{r} from the network, as in (12). Attention must be paid to the initial choice of $p_\alpha(w)$ which still occurs in (11).^e

^eHistorically the choice of prior on the weights has normally been chosen to make the gradients of the likelihood manageable, but this may not be the best justified. Such a choice in prior could be made more meaningful by designing a model where parameters having meaning (see footnote d). Another way to solve this problem is not to consider Bayesian neural networks, but instead transfer the prior distribution of network parameters to the prior distribution of data, $\mathcal{P}(\mathbf{x}, \mathbf{y})$ [4]. Note that, in any case, the prior distribution of data should be considered for a fully Bayesian analysis.

This description of Bayesian neural networks, therefore, refers solely to networks which are part of a Bayesian model,^f i.e. networks where the epistemic uncertainty in the network parameters are characterized by probability distributions, $\rho(w|x, \mathbf{y})$, and thus we are interested in the inference of w . There are several approaches which are effective for characterizing distributions, but each of them have their pros and cons. In Sec. 3, we present some numerically approximate schemes using the exact distributions and some exact schemes using approximate distributions, these fall under the realms of Monte Carlo methods and variational inference.

2.2.1. *Limitations of the Bayesian neural network formulation*

The goal of a Bayesian neural network is to capture epistemic uncertainties. In the absence of any data, the behavior of the model is only controlled by the prior, and should produce large epistemic uncertainties (high variance of the model outputs) for any given input. We then expect that as we update the posterior of network parameters with training data, the epistemic uncertainties should decrease in the vicinity of these training points, as the model is now at least somewhat constrained, but the variance should remain large for Out-Of-Distribution (OOD) regions far from the training set. This is the behaviour that one would expect, however, we want to highlight that nothing in the BNN derivation presented in this section necessarily implies this behaviour in practice.

As in any Bayesian model, the behavior of a Bayesian neural network when data is not constraining is tightly coupled to the choice of prior. However the priors typically used in BNNs are chosen based on practicality and empirical observation rather than principled considerations on the functional space spanned by the neural network.

^fThere is a common misuse of the term Bayesian neural networks to mean networks which predict posterior distributions, say some variational distribution characterized by a neural density estimator for targets, $\ell(\mathbf{y}|x, w)$, but these networks are not providing the true posterior distribution of the target, rather they are simply a fitted distribution approximating (to an unknown degree) the posterior (see Sec. 2.2.2).

There is indeed little guarantee that a Gaussian prior on the weights of a deep dense neural network implies any meaningful uncertainties away from the training distribution. In fact, it is easily shown [4] that putting priors on weights can fail at properly capturing epistemic uncertainties, even on very simple examples.

2.2.2. Relation to classical neural networks

Since neural networks are, in general, able to fit arbitrarily complex models when large enough, we might be able to justify a relatively narrow prior on the hyperparameters, $p(a) \approx \delta(a - \hat{a})$, meaning that we think that an arbitrarily complex network can encapsulate the statistical model $(\mathcal{S}, \mathcal{E}, \mathcal{P})$.^g Marginalizing over the possible hyperparameters gives us

$$\begin{aligned} p(x, y, w) &= \int_{\mathcal{E}_\alpha} da \, p_a(x, y, w) p(a) \\ &= \int_{\mathcal{E}_\alpha} da \, p_a(x, y, w) \delta(a - \hat{a}) \\ &= p_{\hat{a}}(x, y, w). \end{aligned} \quad (13)$$

This describes the probability of possible input-target pairs and network parameters for any given choice of hyperparameters, from which we can write

$$p(y|x, (x, y)_{\text{train}}) = \int_{\mathcal{E}_w} dw \, \hat{\ell}(y|x, w) \hat{\rho}(w|(x, y)_{\text{train}}), \quad (14)$$

where $\hat{\ell} = \ell|_{a=\hat{a}}$ and $\hat{\rho} = \rho|_{a=\hat{a}}$.

In a non-Bayesian context, having restricted the possible forms of neural networks via fixing $a = \hat{a}$, it is common to find the mode of the distribution of neural network parameters, w , by maximizing

^gIn assuming $p(a) \approx \delta(a - \hat{a})$ we are of course neglecting a source of epistemic uncertainty. One possible way that allows us to attempt to characterize the distribution of some subset of a is the use of Bayesian model averaging or ensemble methods [5]. This could be used to sample randomly, for example, from the initialization values of network parameters or the order with which minibatches of data are shuffled, all of which can affect the preferred region of network parameter space which fits the intended function.

the likelihood^h of observing some training set $(\boldsymbol{x}, \boldsymbol{y})_{\text{train}} \in \mathcal{E}$ when given those parameters

$$\hat{\boldsymbol{w}} = \arg \max_{\boldsymbol{w} \in \mathcal{E}_{\boldsymbol{w}}} \hat{\ell}(\boldsymbol{y}_{\text{train}} | \boldsymbol{x}_{\text{train}}, \boldsymbol{w}). \quad (15)$$

Once an estimate for the network parameters is made, the posterior distribution of parameter values, $\hat{\rho}(\boldsymbol{w} | (\boldsymbol{x}, \boldsymbol{y})_{\text{train}})$, is usually degenerated to a delta function at the maximum likelihood estimate of the network parameters, $\hat{\rho}(\boldsymbol{w} | (\boldsymbol{x}, \boldsymbol{y})_{\text{train}}) \Rightarrow \delta(\boldsymbol{w} - \hat{\boldsymbol{w}})$. The prediction of a target, \boldsymbol{y} , from an input, \boldsymbol{x} , then occurs with a probability equal to the likelihood evaluated at the maximum likelihood estimate of the value of the network parameters

$$\begin{aligned} p(\boldsymbol{y} | \boldsymbol{x}, (\boldsymbol{x}, \boldsymbol{y})_{\text{train}}) &= \int_{\mathcal{E}_{\boldsymbol{w}}} d\boldsymbol{w} \hat{\ell}(\boldsymbol{y} | \boldsymbol{x}, \boldsymbol{w}) \hat{\rho}(\boldsymbol{w} | (\boldsymbol{x}, \boldsymbol{y})_{\text{train}}) \\ &= \int_{\mathcal{E}_{\boldsymbol{w}}} d\boldsymbol{w} \hat{\ell}(\boldsymbol{y} | \boldsymbol{x}, \boldsymbol{w}) \delta(\boldsymbol{w} - \hat{\boldsymbol{w}}) \\ &= \hat{\ell}(\boldsymbol{y} | \boldsymbol{x}, \hat{\boldsymbol{w}}). \end{aligned} \quad (16)$$

Once optimized, the form of the distribution chosen to evaluate the training samples, i.e. the loss function, is often ignored and the network output, \boldsymbol{r} , is assumed to coincide with the truth, \boldsymbol{y} . Note, however, that the result of (16) is actually a distribution, characterized by the loss function or a variational distribution, at $\boldsymbol{w} = \hat{\boldsymbol{w}}$, peaked at whatever is dictated by the output of the neural network (and not necessarily the true value of \boldsymbol{y}). Therefore, even in the classical case, we can make an estimation of how likely targets are by evaluating the loss function for different \boldsymbol{y} using frameworks such as Markov methods (described in Sec. 3.1) or fitting the variational distribution for $p(\boldsymbol{y} | \boldsymbol{x}, \boldsymbol{w})$ (described in Sec. 3.2).

^hAs described earlier, an unregularized loss function can be used to evaluate the negative logarithm of likelihood. A regularization term on the network parameters can be added describing our belief in how the weights should behave. In this case the regularized loss is proportional to the negative logarithm of the posterior distribution and maximizing the regularized loss is equivalent to MAP estimation.

However, this form of Bayesian inference does not characterize the uncertainties due to the neural network. Using the maximum likelihood of the network parameters (and hyperparameters) as degenerated prior distributions for calculating the posterior predictive distribution, $p(\mathbf{y}|\mathbf{x}, (\mathbf{x}, \mathbf{y})_{\text{train}})$ completely ignores the epistemic uncertainty introduced by the network by assuming that the likelihood with such parameters exactly describes the distribution of \mathbf{y} given a value of \mathbf{x} . Again, even though the value of $\mathbf{w} = \hat{\mathbf{w}}$ that maximizes the likelihood can be found fairly easily by various optimization schemes, information about the shape of the likelihood is discarded and therefore may not be supported by the bulk of the probability. To incorporate our lack of knowledge and build true Bayesian neural networks, we have to revert back to (12).

3. Practical Implementations

The methods laid out in this chapter showcase some practical ways for characterizing distributions. These distributions could be, for example, the posterior distribution of network parameters, $\rho(\mathbf{w}|\mathbf{x}, \mathbf{y})$, necessary for performing inference with Bayesian neural networks, or likewise, the predictive density of targets from inputs, $p(\mathbf{y}|\mathbf{x})$, normally considered in model inference or, indeed, any other distribution. For simplicity we will refer, abstractly, to the target distribution as

$$\rho(\lambda|\chi) = \frac{\mathcal{L}(\chi|\lambda)p(\lambda)}{e(\chi)} \quad (17)$$

for variables $\lambda \in \mathcal{E}_\Lambda$ and observables $\chi \in \mathcal{E}_X$.

3.1. *Numerically approximate inference: Monte Carlo methods*

Monte Carlo methods define a class of solutions to probabilistic problems. One particularly important method is Markov chain Monte Carlo (MCMC) in which a Markov chain of samples is constructed with such properties that the samples can be attributed as belonging to a target distribution.

A Markov chain is a stochastic model of events where each event depends on *only* one previous event. For example, labeling an event as $\lambda_i \in \mathcal{E}_\Lambda$, the probability of transitioning to another event $\lambda_{i+1} \in \mathcal{E}_\Lambda$ is given by a transition probability, $t : \{\lambda_i, \lambda_{i+1}\} \in \mathcal{E}_\Lambda \mapsto t(\lambda_{i+1}|\lambda_i) \in [0, 1]$, where its value describes how likely λ_{i+1} will be transitioned to from λ_i . A chain consists of a set of events, called samples, of the state, $\{\lambda_i | i \in [1, n]\}$, in which each consecutive sample is correlated with the next. Although the transition probability is only conditional on the previous state, the chains are correlated over long distances. Only states that are physically uncorrelated can be kept as samples from some target distribution, ρ .

One property that a Markov chain must have to represent a set of samples from a target distribution, is *ergodicity*. This means that it is possible to move from any possible state to another in some finite number of transitions from one state to the next and that no long-term repeating cycles occur in the chain. The stationary distribution of the chain, in the asymptotic limit of infinite samples, can be denoted $\pi : \lambda \in \mathcal{E}_\Lambda \mapsto \pi(\lambda) \in [0, 1]$. Since an infinite number of samples are needed to prove the stationary condition, MCMC techniques can only be considered numerical approximations to the target distribution. It should be noted that the initial steps in any Markov chain tend to be out of equilibrium and as such those samples can be out of distribution. All the samples until the stationary distribution is reached are considered *burn-in* samples and need to be discarded in order not to skew the approximated target distribution.

3.1.1. Metropolis–Hastings algorithm

The Metropolis–Hastings algorithm is a method which allows states to be generated from a target distribution, ρ , by defining transition probabilities between states such that the distribution of samples, π , in a Markov chain is stationary and ergodic. This can be ensured easily by invoking *detailed balance*, i.e. making the transition probability from state λ_i to λ_{i+1} reversible such that the Markov chain is necessarily in a steady state. Detailed balance can be written as

$$\pi(\lambda_i)t(\lambda_{i+1}|\lambda_i) = \pi(\lambda_{i+1})t(\lambda_i|\lambda_{i+1}), \quad (18)$$

i.e. the probability of being in state λ_i and transitioning to state λ_{i+1} is equal to the probability of being in state λ_{i+1} and transitioning to state λ_i .

As described in Sec. 3, it can be effectively impossible to characterize a distribution, ρ , since the integral necessary for calculating the marginal for any observed data, $e(\chi)$, can often be intractable. This isn't a problem when using the Metropolis–Hastings algorithm, thanks to detailed balance. First, substituting the target distribution, $\pi(\lambda) \approx \rho(\lambda|\chi)$, into the detailed balance equation and rearranging gives

$$\begin{aligned} \frac{t(\lambda_{i+1}|\lambda_i)}{t(\lambda_i|\lambda_{i+1})} &= \frac{\rho(\lambda_{i+1}|\chi)}{\rho(\lambda_i|\chi)} \\ &= \frac{\mathcal{L}(\chi|\lambda_{i+1})p(\lambda_{i+1})/e(\chi)}{\mathcal{L}(\chi|\lambda_i)p(\lambda_i)/e(\chi)} \\ &= \frac{\mathcal{L}(\chi|\lambda_{i+1})p(\lambda_{i+1})}{\mathcal{L}(\chi|\lambda_i)p(\lambda_i)}. \end{aligned} \quad (19)$$

The intractable integral cancels out and as such we can work with the unnormalized posterior,

$$\begin{aligned} \varrho(\lambda|\chi) &\equiv \mathcal{L}(\chi|\lambda)p(\lambda) \\ &= \rho(\lambda|\chi)e(\chi) \\ &= \mathfrak{p}(\chi, \lambda), \end{aligned} \quad (20)$$

such that

$$\frac{t(\lambda_{i+1}|\lambda_i)}{t(\lambda_i|\lambda_{i+1})} = \frac{\varrho(\lambda_{i+1}|\chi)}{\varrho(\lambda_i|\chi)}. \quad (21)$$

The Metropolis–Hastings algorithm involves breaking the transition probability into two steps, $t(\lambda_{i+1}|\lambda_i) = a(\lambda_{i+1}, \lambda_i)s(\lambda_{i+1}|\lambda_i)$, with a conditional distribution, $s : (\lambda_{i+1}, \lambda_i) \in \mathfrak{E}_\Lambda \mapsto s(\lambda_{i+1}|\lambda_i) \in [0, 1]$, proposing a new sample and a probability, $a(\lambda_{i+1}, \lambda_i)$, describing whether the new sample is accepted as a valid proposal or not.

Substituting these into the detailed balance equations gives

$$\frac{a(\lambda_{i+1}, \lambda_i)}{a(\lambda_i, \lambda_{i+1})} = \frac{\varrho(\lambda_{i+1}|\chi)s(\lambda_i|\lambda_{i+1})}{\varrho(\lambda_i|\chi)s(\lambda_{i+1}|\lambda_i)}. \quad (22)$$

A reversible acceptance probability can then be identified as

$$a(\lambda_{i+1}, \lambda_i) = \min \left[1, \frac{\varrho(\lambda_{i+1}|\chi)s(\lambda_i|\lambda_{i+1})}{\varrho(\lambda_i|\chi)s(\lambda_{i+1}|\lambda_i)} \right], \quad (23)$$

such that either $a(\lambda_{i+1}, \lambda_i) = 1$ or $a(\lambda_i, \lambda_{i+1}) = 1$.ⁱ

The algorithm itself has two free choices, the first is the number of iterations needed to overcome the correlation of states in the chain and properly approximate the target distribution, but in principle it should approach infinity. The second is the choice of the proposal distribution, s . It is often chosen to be a multivariate Gaussian whose covariance can be optimized during burn-in to properly represent useful step sizes in the direction of each element of a state. This ensures that the Markov chain is a random walk. A poor choice of proposal distribution can cause extremely inefficient sampling and as such it should be chosen carefully.

Whilst, in principle, Metropolis–Hastings MCMC will work in high dimensions, the rejection rate can be high and the correlation length very long. Above a handful of parameters, the computational time of Metropolis–Hastings becomes a limitation, meaning that it is not efficient for sampling high-dimensional distributions such as the posterior distribution of neural network parameters.

ⁱWhilst a reversible Markov chain enforces stationarity, it also leads to a probability of rejecting samples, which can be inefficient. Although we will not go into detail here, it is also possible to construct a continuous, directional Markov process which is still ergodic. In this case every sample from the state will be accepted making the algorithm more efficient for collecting samples — although the computation could be more costly. One example of such a method is the Bouncy Particle Sampler [6, 7] in which samples are obtained from the target distribution by picking a random direction in parameter space and sampling along a piecewise-linear trajectory until the value of target distribution at that state is less than or equal to the value of the target distribution at the initial state. At this point there is a Poissonian probability of the trajectory bouncing back along another randomized trajectory, drawing samples along the way. Such methods are state-of-the-art but mostly untested in the literature on sampling neural networks.

3.1.2. Hamiltonian Monte Carlo

One way of dealing with the large correlation between samples, high rejection rate and small step sizes which occur in Metropolis–Hastings is to introduce a new sampling proposal procedure based on a Gibbs sampling step and a Metropolis–Hastings acceptance step. In Hamiltonian Monte Carlo (HMC), we introduce an arbitrary *momentum* vector, ν , with as many elements as λ has. We describe the Markov process as a classical mechanical system with a total energy (Hamiltonian)

$$\begin{aligned}\mathcal{H}(\lambda, \nu) &= \mathcal{K}(\nu) + \mathcal{V}(\lambda) \\ &= \frac{1}{2} \nu^T \mathbf{M}^{-1} \nu - \log \varrho(\lambda | \chi).\end{aligned}\quad (24)$$

$\mathcal{K}(\nu)$ is a *kinetic energy* with a “mass” matrix, \mathbf{M} , describing the strength of correlation between parameters. $\mathcal{V}(\lambda)$ is a *potential energy* equal to the negative logarithm of the target distribution. A state, $\mathbf{z} = (\lambda, \nu)$, in the stationary distribution of the Markov chain, $\pi(\lambda, \nu)$, is a sample from the distribution $\mathbf{p}(\lambda, \nu | \chi) = \exp[-\mathcal{H}(\lambda, \nu)]$, found by solving the ordinary differential equation (ODE) derived from Hamiltonian dynamics

$$\dot{\lambda} = \mathbf{M}^{-1} \nu, \quad (25)$$

$$\dot{\nu} = -\nabla \mathcal{V}(\lambda), \quad (26)$$

where the dots are derivatives with respect to some time-like variable, which is introduced to define the dynamical system. The stationary distribution, $\pi(\lambda, \nu) \approx \mathcal{H}(\lambda, \nu)$, of the Markov chain is separable, $\exp[-\mathcal{H}(\lambda, \nu)] = \exp[-\mathcal{K}(\nu)] \exp[-\mathcal{V}(\lambda)]$, and so $\mathbf{p}(\lambda, \nu | \chi) \propto \rho(\lambda | \chi) \mathbf{p}(\nu)$. This means that a Gibbs sample of the i th momentum can be drawn, $\nu_i \sim \mathbf{p} = \mathbb{N}(\mathbf{0}, \mathbf{M})$, and by evolving the state $\mathbf{z}_i = (\lambda_i, \nu_i)$ using Hamilton’s equations, a proposed sample obtained, $\mathbf{z}_{i+1} = (\lambda_{i+1}, \nu_{i+1}) \sim \mathbf{p}_\chi$, i.e. λ_{i+1} and ν_{i+1} are drawn with probability $\mathbf{p}(\lambda_{i+1}, \nu_{i+1} | \chi)$. The acceptance condition for the detailed balance is obtained by computing the difference in energies between the i th

state and the proposed, $(i + 1)$ th, state

$$a(\mathbf{z}_{i+1}, \mathbf{z}_i) = \min [1, \exp(\Delta\mathcal{H})], \quad (27)$$

where any loss in total energy $\Delta\mathcal{H} = \mathcal{H}(\lambda_{i+1}, \nu_{i+1}) - \mathcal{H}(\lambda_i, \nu_i)$ arises from the discretization of solving Hamilton's equations. If the equations were solved exactly (the Hamiltonian is conserved), then every single proposal would be accepted. It is typical to use ϵ -discretization (the leapfrog method, see algorithm 1) to solve the ODE over a number of steps, L , where ϵ describes the step size of the integrator. Smaller step sizes result in higher acceptance rate at the expense of longer computational times of the integrator, whilst larger step sizes result in shorter integration times, but lower acceptance. It is possible to allow for self-adaptation of ϵ using properties of the chain, such as the average acceptance as a function of iteration, and a target acceptance rate, $\delta \in [0, 1]$. It has been shown that, for HMC, the optimal acceptance rate is $\delta \approx 0.65$ and so we can adapt ϵ to be of this order [8]. Care has to be taken though, since the initial samples in the Markov chain will be out of equilibrium and so adapting ϵ in the early iterations can still lead to poor step size later on, and so this adaptation should only be attempted after the burn-in phase. A priori, it is not known how many steps to take in the integrator and so multiple examples of the HMC may need to be run to tune the value of L , which can be very expensive.^j

No U-turn sampler [10]: A proposed extension to HMC to deal with the unknown number of steps in the integrator is the No U-turn

^jRecent work has been done using neural networks to approximate the gradient of target distribution, $\nabla\mathcal{V}(\lambda)$ [9]. Whilst this could lead to errors if trusted for the whole process, the neural gradients are only used in the leapfrog steps to propose new targets, at which point the true target distribution can be evaluated. In this case, a poorly trained estimator of the gradient of the target distribution proposes poor states, and as such the acceptance rate drops, but the samples obtained are still evaluated from the actual target distribution and therefore it is unbiased by the neural network. Furthermore, any rejected states could be rerun numerically (rather than being estimated) and added to the training set to further fit the estimator, potentially providing exponential speed up as samples are drawn. Note, the gradient of the target distribution could be fit using efficient methods described in Sec. 3.2.

Algorithm 1: Leapfrog algorithm (ϵ -discretization)

Input: Initial state, $\mathbf{z} = (\lambda, \nu)$; number of steps, L ; step size, ϵ ; mass matrix, \mathbf{M} ;

Output: Proposed state, $\mathbf{z} = (\lambda, \nu)$;

Calls: Gradient of target distribution, $\nabla \mathcal{V}(\lambda)$;

$\nu \leftarrow \nu - \epsilon \nabla \mathcal{V}(\lambda)/2$;

for $i \leftarrow 1$ **to** L **do**

$\lambda \leftarrow \lambda + \epsilon \mathbf{M}^{-1} \nu$;

if $i \neq L$ **then**

$\nu \leftarrow \nu - \epsilon \nabla \mathcal{V}(\lambda)$;

end

end

$\nu \leftarrow \nu - \epsilon \nabla \mathcal{V}(\lambda)/2$;

sampler (NUTs). Here, the idea is to find a condition which describes whether or not running more steps in the integrator would carry on increasing the distance between the initial sample and a proposed one. A simple choice of criterion is the derivative with respect to Hamiltonian time of the half squared distance between the current proposed and initial states

$$\begin{aligned} \mathfrak{d} &= \frac{d}{dt} \frac{(\lambda_{i+1} - \lambda_i) \cdot (\lambda_{i+1} - \lambda_i)}{2} \\ &= (\lambda_{i+1} - \lambda_i) \cdot \nu. \end{aligned} \tag{28}$$

If $\mathfrak{d} = 0$ then this indicates that the dynamical system is starting to turn back on itself, i.e. making a U-turn, and further proposals can be closer to the initial state. In practice, a balanced binary tree of possible samples is created by running the leapfrog integrator either forwards or backwards for a doubling number of steps $(1, 2, 4, 8, \dots)$ where each of these steps is a leaf of the tree, $\mathcal{F} = \{(\lambda^{L\pm}, \nu^{L\pm}) | L \in [1, 2, 4, \dots]\}$. When the furthest distance in the trajectory, $\lambda^{\max L+} - \lambda^{\max L-}$, starts to decrease then the computation can be stopped and we can sample from \mathcal{F} via a detailed-balance preserving method. Such an algorithm can greatly reduce the cost of tuning the number

of steps in the integrator, L , in the HMC and is therefore highly beneficial when attempting to characterize a target distribution.

Thanks to the high acceptance rate and the ability to take large steps to efficiently obtain samples, HMC is a good proposition for numerically approximating distributions such as the posterior of neural network parameters. One severe limitation, though, is the choice of the mass matrix, \mathbf{M} . The mass matrix must be properly defined since it defines the direction and size of steps and correlations between parameters. It is not easy to choose its value *a priori* and a poor choice can lead to very inefficient sampling. We present below two methods which deal with the mass matrix.^k

Quasi-Newtonian HMC [12]: With quasi-Newtonian HMC (QNHMC) we make use of the second-order geometric information of the target distribution as well as the gradient. The QNHMC modifies Hamilton's equations to

$$\dot{\lambda} = \mathbf{B}\mathbf{M}^{-1}\nu, \quad (29)$$

$$\dot{\nu} = -\mathbf{B}\nabla\mathcal{V}(\lambda), \quad (30)$$

where \mathbf{B} is an approximation to the inverse Hessian derived using quasi-Newton methods, for more details see [12]. Obtaining this approximation of the Hessian is extremely efficient because all the necessary components are calculated when solving Hamilton's equations using leapfrog methods as in Algorithm 1. Note that the approximate inverse Hessian varies with proposal, but is kept constant whilst solving Hamilton's equations. The inverse Hessian effectively rescales the momenta and parameters such that each dimension has a similar scale and thus the movement around the target distribution is more efficient with less correlated proposals. It is easiest to begin

^kNote we are not going to discuss relativistic HMC [11], where the kinetic energy is replaced with its relativistic form $\mathcal{K}(\nu_j) = \sum_{i=1}^{\dim \lambda} m_i c_i^2 \sqrt{(\nu_j/m_i c)^2 + 1}$. Whilst this method is valid for preventing the run-away of particles on very glassy target distributions thanks to an upper bound on the distance able to be traveled per iteration, m and c are (in practice) needed for every momenta in the dimension of λ . This makes it as difficult a problem as *a priori* knowing the mass matrix, \mathbf{M} , in the classical case.

with an initial inverse Hessian, $\mathbf{B}_0 = \mathbb{I}$, and allow the adaptation of the Hessian to the geometry of the space. Note that the mass matrix, \mathbf{M} , is still present to set the dynamical time-like scales of Hamilton's equations along each direction, but the rescaling of the momenta via \mathbf{B} allows us to be fairly ambiguous about its value. The optimal mass matrix for sampling is equal to the covariance of the target distribution, but in practice, a diagonal mass matrix with approximately correct variance values for the distribution works well.

Example: Inference of the halo mass distribution function:

To be able to extract cosmological information from the large scale-structure distribution of matter in the universe, such as the mass, location and clustering of galaxies, obtained by galaxy surveys, we either have to summarize the data into statistical quantities (such as the power spectrum, etc.) or learn about the placement of all the objects in these surveys. Whilst the first method is (potentially very) lossy, the complexity of the likelihood describing the distribution of structures in the universe generally makes the second technique intractable. With the goal of maximizing the cosmological information extracted from galaxy surveys, the Aquila consortium has developed an algorithm for Bayesian origins reconstruction from galaxies (BORG) [13–15] which assumes a Bayesian hierarchical model to relate Gaussian initial conditions of the early universe to the complex distribution of galaxies observed today. As part of this model, one needs to relate observed galaxies to the underlying, and otherwise invisible, dark matter field through a so-called *bias* model, which is an effective description for extremely complex astrophysical effects. Finding a flexible enough and yet tractable parameterization for this model *a priori* is a difficult task.

Using physical considerations, such as locality and radial symmetry, to reduce the numbers of degrees of freedom, a very simple mixture density network with 17 parameters was proposed to model this bias [3]. This network, dubbed a neural physical engine due to its physical inductive biases, is small enough that each parameter is exactly identifiable, so that sensible priors could be defined for those parameters. The ability to place these meaningful priors on

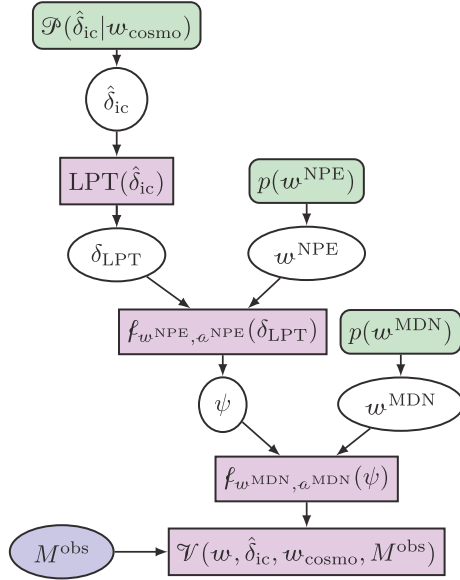


Fig. 1. Schematic of the BORG algorithm with the neural bias model. Initial conditions for the dark matter density field in Fourier space, $\hat{\delta}_{\text{ic}}$, are drawn from a prior given a cosmology w_{cosmo} , $\mathcal{P}(\hat{\delta}_{\text{ic}}|w_{\text{cosmo}})$. These are then evolved forward using a deterministic prescription, in this example using Lagrangian perturbation theory (LPT). The evolved field, δ_{LPT} , is then transformed further using a neural physical engine, $f_{w^{\text{NPE}}, a^{\text{NPE}}}(\delta_{\text{LPT}})$, whose form is described by a^{NPE} and which requires parameters w^{NPE} which are drawn from a prior $p(w^{\text{NPE}})$. This provides a field ψ from which the halo mass distribution can be described using a mixture density network, whose hyperparameters are a^{MDN} , with parameters w^{MDN} drawn from a prior $p(w^{\text{MDN}})$. This halo mass distribution function can be evaluated at given halo masses to be compared to the masses of haloes M^{obs} from an observed halo catalog via a Poissonian likelihood, $\mathcal{V}(w, \hat{\delta}_{\text{ic}}, w_{\text{cosmo}}, M^{\text{obs}})$. The initial phases of the dark matter distribution, $\hat{\delta}_{\text{ic}}$, are sampled using HMC and the parameters of the neural bias model made up of the neural physical engine and the mixture of distributions, $w = (w^{\text{NPE}}, w^{\text{MDN}})$, are sampled using QNHMC. Figure adapted from [3].

network parameters is well motivated for this physically motivated problem, but may be more difficult to design for problems without physical intuition.

Sampling from this model could also be integrated within the larger hierarchical model of the BORG framework using QNHMC (see Fig. 1 for a description of the BORG+neural bias model

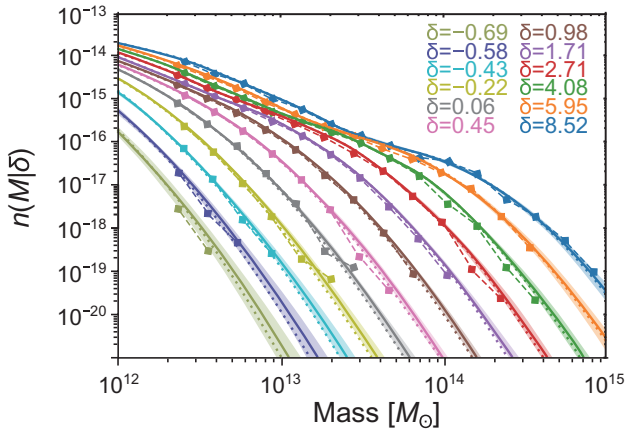


Fig. 2. The halo mass distribution function as a function of mass. The diamonds connected by a dashed line indicates the number density of haloes from an observed halo catalog of a given mass, where the different colors represent the value of the density environment for those haloes. The lines higher in number density correspond to the more dense regions, i.e. there are more large haloes in denser environments. The solid lines show the mean halo number density from samples (taken from the Markov chain) from the neural bias model, with the shaded bands as the 68% credible intervals of these samples. There is a very good agreement between the observed halo number density and that obtained by the neural bias model. Figure credit: Charnock *et al.* (2020) [3].

algorithm). Concretely, BORG was run in two blocks, first using HMC to propose samples of the dark matter density field and then using QNHMC to propose possible neural bias models. The target distribution was assumed to be a Poissonian sampling of the number density of haloes in any particular environment as described by a mixture density network, $\ell_{w^{\text{MDN}}, a^{\text{MDN}}}(\psi)$, evaluated at possible halo masses, \mathbf{m} , and the summarized environmental properties, $\psi = \ell_{w^{\text{NPE}}, a^{\text{NPE}}}(\delta_{\text{LPT}})$, given by the neural physical engine. The parameters in the target likelihood for the masses of haloes in a halo catalog can be explicitly written in terms of the output of the mixture density network, i.e. $\alpha_{\iota, i_h}(\psi(\delta_{\text{LPT}, i_h}, w^{\text{NPE}}), w^{\text{MDN}})$, $\mu_{\iota, i_h}(\psi(\delta_{\text{LPT}, i_h}, w^{\text{NPE}}), w^{\text{MDN}})$ and $\sigma_{\iota, i_h}(\psi(\delta_{\text{LPT}, i_h}, w^{\text{NPE}}), w^{\text{MDN}})$ where i labels the number of voxels in the simulator, h labels the number of halos in the catalog and ι labels the number of Gaussians

in the mixture. The Poissonian likelihood is then written as

$$\begin{aligned}
 \mathcal{V}(\boldsymbol{w}) = & \sum_{h \in \text{catalog}} \log \left[\sum_{\iota}^N \frac{\alpha_{\iota, i_h}}{\sqrt{2\pi\sigma_{\iota, i_h}^2}} \exp \left[-\frac{(\log(\mathbf{m}_h) - \mu_{\iota, i_h})^2}{2\sigma_{\iota, i_h}^2} \right] \right] \\
 & - V \sum_{i \in \text{voxels}} \sum_{\iota}^N \frac{\alpha_{\iota, i}}{2} \exp \left[\frac{\sigma_{\iota, i}^2}{2} \right] \\
 & \times \text{erfc} \left[\frac{\log(\mathbf{m}_{\tau}) - \mu_{\iota, i} - \sigma_{\iota, i}^2}{\sqrt{2\sigma_{\iota, i}^2}} \right], \tag{31}
 \end{aligned}$$

where \mathbf{m}_h is the mass of halo h , \mathbf{m}_{τ} is a threshold on the minimum mass of halos and V is the volume of a single voxel. Using this, the exact joint posterior of both density field and network parameters could be inferred under the observation of a mock halo catalog from the VELMASS simulations [16]. Such inference was an example of zero-shot training since there was no *training* data necessary. Because quasi-Newtonian HMC was used then the correlation between all of the network parameters, $\boldsymbol{w} = (\boldsymbol{w}^{\text{NPE}}, \boldsymbol{w}^{\text{MDN}})$, could be assumed to be negligible, i.e. $\mathbf{M} = \mathbf{I}$. Whilst this assumption is incorrect, the approximately calculated Hessian, \mathbf{B} , rescaled the parameters such that their correlations were taken into account so that the proposed samples in the HMC could successfully travel along preferred parameter directions. The methods presented here provide a way to constrain the initial phases of the dark matter distribution, conditional on the observed data, without an exact physical description of how the observed catalog trace the underlying dark matter distribution today. This is possible since we can use the neural bias model to map from the dark matter distribution to some unknown function that describes how a catalog of observed halos traces the underlying dark matter based on some physical principles. These physical principles are built directly into the neural physical engine. Any uncertainty in the form of this description can then be marginalized out since the distribution of the parameters in the neural bias model is available via the samples obtained in the QNHMC.

Riemannian Manifold HMC [17]: Whilst we have so far depended on a choice of mass matrix to set the time-like steps in the integrator, it is possible to exploit the geometry of the Hamiltonian to adaptively avoid having to choose. Samples from the Hamiltonian are effectively points in a Riemannian surface with a metric defined by the Fisher information of the target distribution, $\mathcal{J}(\lambda) = \langle \nabla \varrho(\lambda'|\chi) (\nabla \varrho(\lambda'|\chi))^T \rangle_\lambda$. Since the Fisher information describes the amount of information a random observable, χ , contains about any model parameters λ , then the parameter space has larger curvature wherever there is a lot of support from the data. In essence, this metric is a position-dependent equivalent to the mass matrix which we have so far considered, but since we have to calculate $\nabla \varrho(\lambda|\chi)$ in the integrator anyway, we can actually approximate the Fisher information cheaply. However, to ensure that the Hamiltonian is still the logarithm of a probability density it must be regularized leaving the Hamiltonian as

$$\begin{aligned} \mathcal{H}(\lambda, \nu) &= \mathcal{K}(\lambda, \nu) + \mathcal{V}(\lambda) \\ &= \frac{1}{2} \nu^T \mathcal{J}(\lambda)^{-1} \nu - \log \varrho(\lambda|\chi) + \frac{1}{2} \log(2\pi)^{\dim \lambda} |\mathcal{J}(\lambda)|. \end{aligned} \quad (32)$$

Here the kinetic term now involves a dependence on the parameters, λ , and so the Hamiltonian is not separable, i.e. the momenta are drawn from a parameter-dependent mass matrix, $\nu \sim \mathcal{N}(\mathbf{0}, \mathcal{J}(\lambda))$. The equations of motion in this case become

$$\dot{\lambda} = \mathcal{J}(\lambda)^{-1} \nu, \quad (33)$$

$$\begin{aligned} \dot{\nu} &= -\nabla \mathcal{V}(\lambda) + \frac{1}{2} \text{Trace} [\mathcal{J}(\lambda)^{-1} \nabla \mathcal{J}(\lambda)] \\ &\quad - \frac{1}{2} \nu^T \mathcal{J}(\lambda)^{-1} \nabla \mathcal{J}(\lambda) \mathcal{J}(\lambda)^{-1} \nu. \end{aligned} \quad (34)$$

With such a change, the scaling of the momenta along each parameter direction becomes automatic, but the reversibility and volume preserving evolution using the leapfrog integrator is broken and so the proposed states do not adhere to detailed balance. Instead a new symplectic integrator, i.e. an integrator for Hamiltonian systems, is required to make the volume preserving transformation of the

momenta (by calculating the Jacobian of the inverse Fisher matrix) such that Hamilton's equations can be solved. Whilst this adds extra complexity to the equations of motion, it is equivalent to only two additional steps in the integrator since the Fisher information can be approximated cheaply from the calculation of the gradient of the potential energy. By using the RMHMC, we avoid the need to choose a mass matrix (or approximate the Hessian). ϵ can be fixed to some value as the adaptive matrix is able to overcome the step size, and L , i.e. the number of steps in the integrator, can be chosen to tune the acceptance rate.

Stochastic gradient HMC [18]: Whilst we can sample effectively using Hamiltonian Monte Carlo, and its variants shown above, we must also address the question of the size of the data we are interested in. As so far presented, HMC requires an entire dataset, $\chi \in \mathcal{E}_X$, to evaluate the distribution. However, in modern times, datasets can be extremely large (big data) and it may not be possible to evaluate it all simultaneously. Furthermore, we are more and more likely to be in the regime where the data is obtained continually and streamed for inference. For this reason, most optimization techniques rely on stochastic estimation techniques over minibatches, χ_i i.e. the union of all minibatches is the complete set, $\bigcup_i^{\text{batches}} \chi_i = \chi$. This stochastic sampling of data can be considered as being equivalent to adding a source of random noise, or scatter, around the target distribution. For clarity, the gradient of the potential energy for a minibatch becomes

$$\begin{aligned} \nabla \mathcal{V}(\lambda) &= -\log \varrho(\lambda|\chi_i) \\ &\rightarrow -\log \varrho(\lambda|\chi) + \gamma, \end{aligned} \tag{35}$$

where $\gamma \sim \text{Dist}(\lambda)$ is a random variable drawn from the distribution of noise and whose shape is described by some diffusion matrix, $\mathbb{Q}(\lambda)$. For large minibatch sizes, we can relatively safely assume this distribution is Gaussianly distributed, $\gamma \sim \mathcal{N}(\mathbf{0}, \epsilon \mathbb{Q}(\lambda)/2)$, due to the central limit theorem where the diffusion matrix at any step in the integration can be equated to the variance of the noise,

$\epsilon \mathbb{Q}(\lambda) = 2\Sigma(\lambda)$. Note that we may not necessarily be in the regime where we can make this assumption.

Making noisy estimates of the target distribution using mini-batches breaks the Hamiltonian dynamics of the HMC and as such extremely high rejection rates can occur. In particular, the additional noise term acts as force which can push the states far from the target distribution. We can reduce this effect by taking further inspiration from mechanical systems — we can use Langevin dynamics to describe the macroscopic states of a statistical mechanical system with a stochastic noise term describing the expected effect of some ensemble of microscopic states. In particular, using second-order Langevin equations is equivalent to including a friction term which decreases the energy and, thus, counterbalances the effect of the noise. Equations (25) and (26) therefore get promoted to

$$\dot{\lambda} = \mathbf{M}^{-1}\nu, \quad (36)$$

$$\dot{\nu} = -\nabla \mathcal{V}(\lambda) - \mathbb{Q}(\lambda)\mathbf{M}^{-1}\nu + \gamma. \quad (37)$$

Solving these equations provides a stationary distribution, $\pi(\lambda, \nu) \approx \mathcal{H}(\lambda, \nu)$, with the distribution of samples, $\lambda \sim \rho_\chi$, i.e. λ is drawn with probability $\rho(\lambda|\chi)$. Of course, this method depends on knowing the distribution of the noise well, but for large minibatch sizes, this approaches Gaussian. The stochastic gradient HMC, in this case, provides a way to obtain samples from the target distribution even when not using the entire dataset and therefore vastly reducing computational expense and allowing for active collection and inference of data.

3.2. Variational inference

Whilst a target probability distribution can be approximately characterized by obtaining exact samples from the distribution via Monte Carlo methods, it is often a very costly process. Instead we can use variational inference, where a *variational* distribution, say $q^m(\lambda|\chi)$, is chosen to represent a very close approximation to the target

distribution, $\rho(\lambda|\chi)$. In general, $q^m(\lambda|\chi)$ is a tractable distribution, parameterized by some m , and via the optimization of these parameters $q^m(\lambda|\chi)$ can hopefully be made close to the target distribution, $\rho(\lambda|\chi)$. Note, again, that if the target distribution is the posterior predictive distribution of some model, $p(y|x)$, then fitting a variational distribution to this is not a Bayesian procedure in the same way that maximum likelihood estimation is not Bayesian.

To describe what is meant by *close* in the context of distributions, we often consider the Kullback–Leibler (KL) divergence (or relative entropy). In a statistical setting, the KL-divergence is a measure of information lost when approximating a distribution $\rho(\lambda|\chi)$ with some other $q^m(\lambda|\chi)$,

$$\mathbb{KL}(\rho||q^m) = \int_{\mathcal{E}_\Lambda} d\lambda \rho(\lambda|\chi) \log \frac{\rho(\lambda|\chi)}{q^m(\lambda|\chi)}. \quad (38)$$

When $\mathbb{KL}(\rho||q^m) = 0$, there is no information loss and so $\rho(\lambda|\chi)$ and $q^m(\lambda|\chi)$ are equivalent. Values $\mathbb{KL}(\rho||q^m) > 0$ indicate the degree of information lost. Note that the KL-divergence is not symmetric and as such is not a real distance metric.

The form of (38) assumes the integral of $\rho(\lambda|\chi)$ to be tractable. In fact, in the case that the expectation can be approximated well, we can use the KL-divergence to perform expectation propagation. However, if we are considering the approximation of the posterior distribution of network parameters, as stated in Sec. 2, we can expect the integral of $p_a(w|x, y)$ to be intractable meaning that calculating $\mathbb{KL}(p_a||q^m)$ would be necessarily hard. Instead we can consider the reverse KL-divergence

$$\mathbb{KL}(q^m||\rho) = \int_{\mathcal{E}_\Lambda} d\lambda q^m(\lambda|\chi) \log \frac{q^m(\lambda|\chi)}{\rho(\lambda|\chi)}. \quad (39)$$

The choice of $q^m(\lambda|\chi)$ is specified so that expectations are tractable. However, evaluating $\rho(\lambda|\chi)$ would require calculating the evidence, $e(\chi)$, which, although constant for different λ , remains intractable. For convenience we can consider the unnormalized distribution (as

we did for detailed balance in Sec. 3.1.1)

$$\begin{aligned}\varrho(\lambda|\chi) &= \mathcal{L}(\chi|\lambda)p(\lambda) \\ &= \rho(\lambda|\chi)e(\chi) \\ &= \mathfrak{p}(\chi, \lambda),\end{aligned}\tag{40}$$

and calculate a new measure

$$\text{ELBO}(\mathfrak{q}^m) = - \int_{\mathfrak{E}_\Lambda} d\lambda \mathfrak{q}^m(\lambda|\chi) \log \frac{\mathfrak{q}^m(\lambda|\chi)}{\varrho(\lambda|\chi)}.\tag{41}$$

Note that this has the form of minus the reverse KL-divergence, but is not equivalent since $\varrho(\lambda|\chi)$ is not normalized. By substitution, we can see that

$$\begin{aligned}\text{ELBO}(\mathfrak{q}^m) &= - \int_{\mathfrak{E}_\Lambda} d\lambda \mathfrak{q}^m(\lambda|\chi) \log \frac{\mathfrak{q}^m(\lambda|\chi)}{\rho(\lambda|\chi)e(\chi)} \\ &= - \int_{\mathfrak{E}_\Lambda} d\lambda \mathfrak{q}^m(\lambda|\chi) \log \frac{\mathfrak{q}^m(\lambda|\chi)}{\rho(\lambda|\chi)} + \log e(\chi) \\ &= -\mathbb{KL}(\mathfrak{q}^m||\rho) + \log e(\chi).\end{aligned}\tag{42}$$

Since $\log e(\chi)$ is constant with respect to the parameters, λ , maximizing $\text{ELBO}(\mathfrak{q}^m)$ will force $\mathfrak{q}^m(\lambda|\chi)$ close to the target distribution $\rho(\lambda|\chi)$. The term ELBO comes from the fact that the KL-divergence is non-negative and so $\text{ELBO}(\mathfrak{q}^m)$ defines a lower bound to the evidence, $e(\chi)$.

3.2.1. Mean-field variation

One efficient way of parameterizing a distribution for approximating a target, $\rho(\lambda|\chi)$, is to make it factorize along each dimension of the parameters, i.e.

$$\mathfrak{q}^m(\lambda|\chi) = \prod_{i=1}^{\dim \lambda} \mathfrak{q}_i^m(\lambda_i|\chi).\tag{43}$$

In doing such, the ELBO for any individual q_j^m is

$$\begin{aligned}
 \text{ELBO}(q_j^m) &= \int \cdots \int_{\mathcal{E}_{\Lambda,j}} d\lambda_j \prod_i q_i^m(\lambda_i|\chi) \\
 &\quad \times \left[\log \varrho(\lambda|\chi) - \sum_k \log q_k^m(\lambda_k|\chi) \right] \\
 &= \int_{\mathcal{E}_{\Lambda,j}} d\lambda_j q_j^m(\lambda_j|\chi) \int \cdots \int_{\mathcal{E}_{\Lambda,i \neq j}} d\lambda_i \prod_{i \neq j} q_i^m(\lambda_i|\chi) \\
 &\quad \times \left[\log \varrho(\lambda|\chi) - \sum_k \log q_k^m(\lambda_k|\chi) \right] \\
 &= \int_{\mathcal{E}_{\Lambda,j}} d\lambda_j q_j^m(\lambda_j|\chi) \\
 &\quad \times \left[\mathbb{E}_{i \neq j} [\log \varrho(\lambda_j|\chi)] - \log q_j^m(\lambda_j|\chi) \right] + \text{const},
 \end{aligned} \tag{44}$$

where the constant is the expectation value of the factorized distributions, $q_i^m(\lambda_i|\chi)$, in the dimensions where $i \neq j$ and is unimportant for the optimization of the distribution for the j th dimension since it is independent of λ_j . $\mathbb{E}_{i \neq j} [\log \varrho(\lambda_j|\chi)]$ is the expectation value of the logarithm of the target distribution for every $q_i^m(\lambda_i|\chi)$ where $i \neq j$, and remains due to its dependence on λ_j . The optimal j th distribution is the one that maximizes the ELBO which is equivalent to optimizing each of the factorized distributions, $q_j^m(\lambda_j|\chi)$, in turn to obtain $q_j^m(\lambda_j|\chi) = \exp [\mathbb{E}_{i \neq j} [\log \varrho(\lambda_j|\chi)]]$. This provides a mean-field approximation of the target distribution.

3.2.2. Bayes by Backprop

Bayes by Backprop [19, 20] (a form of stochastic gradient variational Bayes) provides a method for approximating a target distribution, $\rho(\lambda|\chi)$, using differentiable functions such as neural networks. The basic premise of Bayes by Backprop relies on a technique known as

the *reparameterization trick* [21, 22]. The reparameterization trick provides a method of drawing random samples, w , from a Gaussian distribution, whilst allowing the samples to be differentiable with respect to the parameters of the Gaussian distribution (mean and standard deviation, $m = (\mu, \sigma)$). This can be achieved by reparameterizing $\mathcal{N}(\mu_i, \sigma_i)$ in terms of an independent normally distributed auxiliary variable, $\epsilon \sim \mathcal{N}(0, 1)$, i.e.

$$\begin{aligned} w(m_i) &\sim \mathcal{N}(\mu_i, \sigma_i) \\ &= \mu_i + \sigma_i \epsilon_i. \end{aligned} \quad (45)$$

Here, we can view $w(m_i)$ as the i th random variable parameter of a neural network with a total of n_w network parameters where $w(m) = \{w(m_i) | i \in [1, n_w]\}$. Any evaluation of the neural network is a sample $\hat{\lambda} \sim q_{\chi, w}^m$, i.e. $\hat{\lambda}$ is drawn with probability $q^m(\lambda | \chi, w)$. Maximizing the ELBO (42), between $q^m(\lambda | \chi, w)$ and an unnormalized target distribution, $\varrho(\lambda | \chi)$, can now be done via backpropagation since we can calculate

$$\partial_m \text{ELBO}(q^m) = \left(\begin{aligned} &\partial_{w(m)} \text{ELBO}(q^m) + \partial_\mu \text{ELBO}(q^m) \\ &(\epsilon/\sigma) \partial_{w(m)} \text{ELBO}(q^m) + \partial_\sigma \text{ELBO}(q^m) \end{aligned} \right) \quad (46)$$

and update the parameters using

$$m \leftarrow m - \eta \partial_m \text{ELBO}(q^m), \quad (47)$$

where η is a learning rate. Note that the $\partial_{w(m)} \text{ELBO}(q^m)$ terms in (47) are exactly the same as the gradients normally associated with backpropagation in neural networks.

As originally presented, Bayes by Backprop was an attempt to make Bayesian posterior predictions of targets, y , from inputs, x , as in (12), where $\rho(w | x_{\text{train}}, y_{\text{train}}) \equiv \prod_{i=1}^{n_w} \mathcal{N}(\mu_i, \sigma_i)$. Here, the values of all μ_i and σ_i are fit using maximum likelihood estimation (or maximum *a posteriori* estimation) given data x_{train} and y_{train} . As explained in Sec. 2.2, the distribution of weights, $p_a(w)$, is likely to be extremely non-trivial, since most network parameters are non-identifiable and highly degenerate with other parameters. Therefore,

modeling this distribution as a Gaussian is unlikely to be very accurate. This can, therefore, incorrectly conflate the epistemic uncertainty for \mathbf{y} from a particular network, \mathcal{a} , with parameters, \mathbf{w} , and input, \mathbf{x} , with the posterior prediction. In essence, Bayes by Backprop provides a way of sampling from a single choice of an (arbitrarily complex) approximation of a target distribution much more efficiently than using numerical schemes such as Markov methods, but there is little knowledge in how close this approximation is to the desired target. By fitting the parameters, \mathbf{m} , of the neural distribution rather than characterizing their distribution, $p(\mathbf{m})$, the characterization of the epistemic uncertainty is biased. As such, just using Bayes by Backprop provides a network that is Bayesian in principle, but with a limited choice of prior distribution which may fail to capture our epistemic uncertainty.

Whilst Bayes by Backprop allows us to characterize the mean and standard deviation of a Gaussian distribution from which we can draw network parameters, it can be extremely expensive to draw different parameters for each example of the data to perform the optimization. Therefore, the data is often split into n_{batches} minibatches of n_{elems} elements and a single sample of each parameter drawn for all n_{elems} elements in each minibatch. This clearly does not represent the variability of the distribution of parameters well and leads to artificially high variance in the stochastic gradient calculation. Furthermore, by sharing the same parameter values for all elements in a minibatch, correlations between gradients prevent the high variance from being eliminated.

Example: Classification of photometric light-curves: Because Bayes by Backprop can be comparatively more expensive than other practical techniques introduced below, there are a fairly limited number of examples of applications in the physics literature. One notable example however is the probabilistic classification of supernova lightcurves method **SuperNNova** [23]. The aim of that study is to analyze time-series measuring the brightness of distant galaxies as a function of time, and detect potential supernova type Ia events, of particular interest for cosmology.

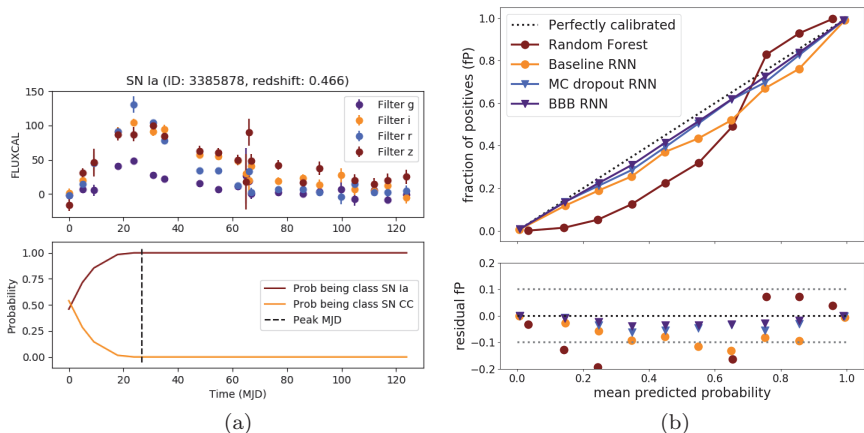


Fig. 3. (a) Illustration of the photometric lightcurve classification problem on simulated supernova type Ia event. Top: Observed flux as a function of time, in different bands (broad wavelength filters). Bottom: Classification probability output from a recurrent neural network (RNN) as a function time. While uncertain about the type of the event (Ia or CC) at the beginning of the event, the model starts recognizing a Ia event and classifying it as such as the event unfolds. Figure credit: Möller and de Boissiere (2019) [23] (b) Calibration of predicted class probabilities for the supernova lightcurve classification problem. The BBB RNN (purple triangle) exhibits better calibration than a vanilla RNN (orange dot) of matching architecture but optimized by maximum likelihood. Figure credit: Möller and de Boissiere (2019) [23].

Figure 3(a) illustrates the light curve classification problem on a simulated supernova type Ia event. In that study, the authors aim to compare the performance of a vanilla recurrent neural network (RNN) classifier to a probabilistic model quantifying some uncertainties. For that purpose the authors introduce a “*Bayesian*” recurrent neural network [24] (BRNN) based on a bidirectional LSTM, but using a variational Gaussian distribution for the posterior of network parameters, optimized by backpropagation. Following the approach presented in this section, the loss function for this model becomes

$$\Lambda(m) = - \mathbb{E}_{w \sim q^m} [\log \ell(\mathbf{y}_{\text{train}} | \mathbf{x}_{\text{train}}, w)] + \text{KL}(q^m || \rho), \quad (48)$$

which corresponds to the ELBO introduced in Eq. (42), and where $\log \ell(\mathbf{y}_{\text{train}} | \mathbf{x}_{\text{train}}, w)$ is the log-likelihood of a categorical distribution

with probabilities predicted by the neural network, and $p_a(w)$ is the prior on the BRNN parameters.

With this approach, the authors attempt to distinguish between aleatoric uncertainties which are uniquely determined by the categorical probabilities predicted by the model for a given set of network parameters w , and the epistemic uncertainties which are, this case, characterized by the variational approximation to the posterior of network parameters $q^m(w|(x, y)_{\text{train}}) \approx \rho(w|(x, y)_{\text{train}})$. As highlighted multiple times before, one should however always be careful in interpreting these probabilities, and in that study the authors empirically check the calibration of the mean posterior probabilities using a reliability diagram [25]. The reliability diagram shows the fraction of true positives in a binary classification problem as a function of the probabilities predicted by the model. For a perfectly calibrated classifier, only 10% of objects which received a detection probability of 0.1 are true positives.

Figure 3(b) shows this calibration diagram for different classifiers, but of particular interest are the curves for the *baseline RNN* and *BBB RNN*, in both cases the actual neural network architecture is identical, but the former is optimized to find the maximum likelihood estimates of the network parameters, while the later is trained by Bayes by Backprop. The *BBB RNN* predicted probabilities are closer to the diagonal representing a more *correct* calibration than the baseline RNN. In this example including a model for the epistemic uncertainties improves the model calibration.

3.2.3. Local reparameterization trick

Although for large numbers of network parameters, n_w , characterizing the global uncertainty of the parameters using the reparameterization trick becomes computationally unfeasible for each element of data and for each parameter in the network, a local noise approximation [26] can be made to transform the perturbation of parameters to a perturbation of activation values

$$o_{jn}^l \sim \mathbb{N} \left(\sum_{i=1}^{\dim l-1} \mu_{ji}^l a_{in}^{l-1}, \sum_{i=1}^{\dim l-1} (\sigma_{ji}^l)^2 (a_{in}^{l-1})^2 \right), \quad (49)$$

where $a_{jn}^l = \ell(o_{jn}^l)$ is the activated output (with possibly nonlinear activation function ℓ) of the j th unit of the l th layer of a neural network with n_l layers, according to the n th element of the input minibatch. As with the reparameterization trick, the sampling of the activation value of any layer can be written as

$$o_{jn}^l = \sum_{i=1}^{\dim l-1} \mu_{ji}^l a_{in}^{l-1} + \epsilon_{jn}^l \sqrt{(\sigma_{ji}^l)^2 (a_{in}^{l-1})^2}, \quad (50)$$

where $\epsilon_{jn}^l \sim \mathcal{N}(0, 1)$. Whilst the parameters $m_{ji}^l = (\mu_{ji}^l, \sigma_{ji}^l)$ of the Gaussian distribution describe the probabilistic model for a network parameter, $w(m_{ji}^l)$, from unit i of layer $l-1$ to unit j of layer l , this model is never sampled, and only the activation values are sampled. The dimensionality of the probabilistic interpretation of layer outputs, i.e. the number of $\epsilon = \{\epsilon_{in}^l | i \in [1, \dim l], l \in [1, n_l], n \in [1, n_{\text{elems}}]\}$ needed to be stored for computation of the gradient is much lower than when considering the number of random draws needed for every single network parameter and every element of data in the minibatch. Furthermore, the variance of the gradient is much less when using the local reparameterization trick than when assuming a single random draw for each parameter being the same for all of the elements of data in a minibatch.

3.2.4. Variational dropout

One limitation of the local reparameterization trick is that it only applies to networks with no weight sharing, i.e. fully-connected neural networks. However, inspired by the local reparameterization trick, a general method for approximating distributions using multiplicative noise can be implemented. Variational dropout [27, 28] is another way of approximating a target distribution, $\rho(\lambda|\chi)$ with $q^m(\lambda|\chi, w)$. In this case, the distribution is defined by the application of random variables to the outputs of hidden layers in a neural network. Much like the local reparameterization trick, the outputs of the n_l layers of a neural network are draws from some multiplicative noise model, $a_{in}^l = \epsilon_{in}^l \ell(o_{in}^l)$, where o_{in}^l are the non-activated outputs of the l th layer of a neural network at element n in the minibatch. Note that

σ_{in}^l can be obtained using any function, i.e. fully connected, convolutional, etc. \mathcal{f} is some (possibly nonlinear) activation function and $\epsilon_{in}^l \sim \text{Dist}(m_{in}^l)$ is a random variable drawn from some distribution parameterized by some $m = \{m_{in}^l | i \in [1, \dim l], l \in [1, n_l], n \in [1, n_{\text{elems}}]\}$. Selecting some form for the distribution and values for its parameters, m , provides a way of obtaining samples from the neural network by running the network forward with many draws of ϵ . This makes the network a model of a Bayesian neural network rather than a Bayesian neural network itself — there is no sampling of the parameters of the network, and no attempt to characterize their uncertainty. Furthermore, the value of m cannot be fit using Bayes by Backprop and it is an *a priori* choice for the sampling distribution.¹

Bernoulli dropout: One method of performing variational dropout is by using $\epsilon \sim \text{Bernoulli}(m)$, which amounts to feeding forward an input to a network with dropout [29] with a keep rate m for each of the outputs of each layer of the neural network multiple times. The outputted samples can then be interpreted as the distribution of possible targets which can be obtained using that network (and the choice of the Bernoulli distribution with parameters m). It is very common to set all values of $m_{in}^l \in m$ to the same value, although it can be optimized via expectation maximization. The ease with which this method can be implemented has made it very popular, and in the limit of large number of samples, the activated outputs approach a Gaussian distribution thanks to the central limit theorem. Note that the choice of a Bernoulli distribution changes the expected output of any activation layer as $\langle a_{in}^l \rangle = m_{in}^l (1 - m_{in}^l) a_{in}^l$, therefore there is a scaling which needs to be taken into account.

Gaussian dropout: A second option is to draw the random variable from a unit-mean Gaussian [26, 30], $\epsilon \sim \mathcal{N}(\mathbb{I}, \text{diag}(m))$, so that the expectation value of the multiplication of the output of a unit of a layer by the random variable remains, $\langle a_{in}^l \rangle = a_{in}^l$ the same since

¹It should be noted that the parameters, m , of any variational dropout distribution can be optimized via expectation maximization.

$\langle \epsilon \rangle = 1$. Furthermore, by calculating the variational objective the value of m in the multiplicative noise distribution can be fit using expectation maximization.

For both the Bernoulli, Gaussian or any other multiplicative dropout distribution, by maximizing the $\text{ELBO}(q^m)$, we can get $q^m(\lambda|\chi, w)$ close to $\rho(\lambda|\chi)$ allowing us to make estimates of this distribution. Again it should be stated that this is not Bayesian in the sense that if the variational distribution provided by variational dropout is approximating the posterior predictive distribution, $p(y|x)$, there is no sense of certainty in how good that approximation is. There is no attempt to characterize our lack of knowledge of the parameters of the network *or* the parameters of the distributions, m .^m

3.2.5. Monte Carlo dropout

Very closely related to Bernoulli variational dropout, is the MC Dropout model [27]. Similar to the previous section, MC Dropout provides a Bayesian framework to interpret the effect of traditional dropout [29] on neural networks. A variational distribution $q^m(w|\chi, \lambda)$ assumed for the network parameter posterior can be parameterized as $w = \mathbf{M} \cdot \text{diag}([z_j]_{j=1}^J)$ with $z_j \sim \text{Bernoulli}(m)$, \mathbf{M} being a $K \times J$ weight matrix, and m being the dropout rate. Given this formulation for the variational distribution it can be shown that a KL divergence with respect to an implicit prior can be approximated as a simple ℓ_2 regularisation term [27]. Training a neural network under dropout and with ℓ_2 weight regularization therefore maximizing the $\text{ELBO}(q^m)$ and is performing proper variational inference at no extra cost.

Example: Probabilistic classification of galaxy morphologies and active learning: MC dropout is the most frequent solution adopted for probabilistic modeling using neural networks, and was

^mOf course, if considering a MAP estimate, then some characterization of our lack of knowledge is taken into account, but the distribution is still neglected.

the first such application in astrophysics [31], for a strong gravitational lensing parameter estimation problem. To illustrate the method and its applications on a more recent example [32], we will consider the problem of classifying galaxy types from cutout images. In the context of modern large galaxy surveys, the challenge is to be able to automatically determine galaxy morphological types without (or with minimal) human visual inspection.

Such a study is based on the result of a large citizen science effort asking volunteers to answer a series of questions to characterize the type and morphology of a series of galaxy images. The task for the neural network is to predict volunteer responses for some galaxy types of particular interest, k . These answers are modeled using a binomial distribution, $\text{Bin}(\nu, N)$, where ν is the probability of a volunteer providing a positive response, and N the number of volunteers asked to answer the question. Based on this model, a probabilistic prediction model can be built from a neural network estimating the parameter ν from a given image x :

$$\Lambda = -\log \text{Bin}(k|x, w, a, N) + \lambda \|w\|_2^2. \quad (51)$$

Figure 4 illustrates the difference between posterior predictions from the fitted model with and without sampling of network outputs via MC dropout. For any given fixed realization (central column), the distribution of predictions is generally over confident, leading to apparent mis-calibration as two out of the seven examples appear to give very low probability to the actual value (second and fifth rows). On the contrary, after sampling from the multiplicative noise distribution (right column), the mean model approximate posterior (green) is significantly wider *and* also can exhibit more complex shapes than a simple binomial distribution. The high variance on the posterior predictions indicates that epistemic uncertainties are more significant, and the authors further measure empirically a much improved, but not perfect, calibration of the MC dropout posterior.

Recognizing that the uncertainties modeled by a choice in probabilistic network are not perfect, the authors still propose an excellent use case for them, in the form of active learning. In the active scenario, approximate and fast inference is preferred over more exact,

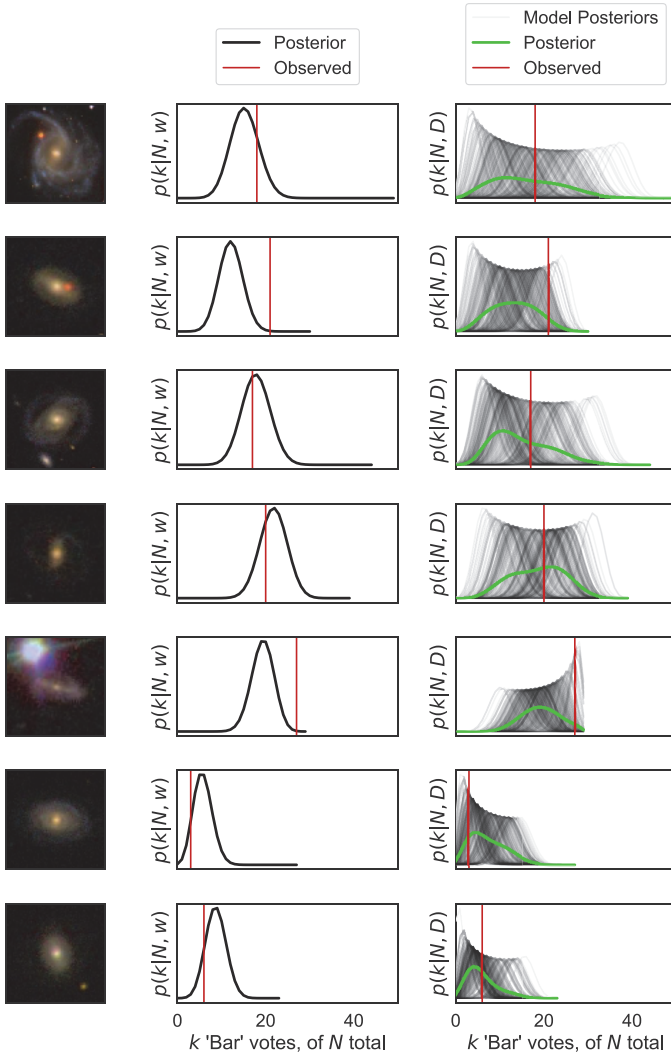


Fig. 4. Posterior distributions of number k of positive answers to the question “Does this image contain a barred spiral galaxy?” for N votes. Left: Image cutouts as presented to the citizen scientists and CNN. Center: Approximate posterior distribution predicted by one model at fixed network parameters, i.e. only modeling aleatoric uncertainties, the red line represents the true observed number. Right: Approximate posterior taking into account a model for both aleatoric and epistemic uncertainties, i.e. obtained by sampling 30 realizations from the MC dropout network. The full posterior distribution (green) is generally broader and better calibrated than individual approximate posterior samples (black). Figure credit: Walmsley *et al.* (2019) [32].

but computationally and time expensive results. For this reason, the authors propose a strategy to identify galaxies for which the model uncertainties are largest, and preferentially ask human volunteers to label those, as a way to selectively invest human resources where they will be the most useful to help constrain the model.

In particular, the authors adopt the Bayesian Active Learning by Disagreement [33] (BALD) strategy which is based on selecting examples that maximize the mutual information $\mathbb{I}[k, \mathbf{w}]$. This quantity measures, for a given galaxy \mathbf{x} , how much information can be gained on the parameters of the neural network \mathbf{w} from knowing the true label k of that galaxy. While estimating this mutual information is in general a difficult task, a practical estimator can be derived in the case of a MC dropout. In their experiments, it is found that for some prediction tasks, as much as 60% fewer training galaxies are necessary to reach a given testing score when selected through active learning, compared to selected through a uniform random sampling.

3.2.6. *Flipout*

Flipout [34] is an alternative to the local reparameterization trick (or variational dropout) that proposes an efficient way to generate (and store) pseudo-independent perturbations to decorrelate the gradients with respect to parameters $\mathbf{m} = \{(\mu_i, \Delta \mathbf{w}_i) | i \in [1, n_w]\}$ according to each of the n_{elems} elements of data within a minibatch. μ_i and $\Delta \mathbf{w}_i$ are the mean and stochastic perturbation of some network parameter $\mathbf{w}(m_i)$. This method therefore is more closely akin to stochastic gradient variational Bayes, where the distribution of network parameters is fitted. Note that for Flipout, the requirement on the distribution for each network parameter is that they are differentiable with respect to the parameters \mathbf{m} , that the distribution of perturbations of the network parameters, $\Delta \mathbf{w}_i$, is symmetric about zero (but not necessarily Gaussian) and the network parameters are independent. With $\Delta \mathbf{w}_i$ symmetric amount zero, the multiplication by a random matrix of signs leaves it identically distributed. This means that by choosing a single $\Delta \mathbf{w}_i$ for each network parameter (like $\Delta \mathbf{w}_i = \sigma_i \epsilon_i$ for the Gaussian case) somewhat decorrelated gradients can be obtained for each element of a minibatch by identically

distributing Δw_i via random draws from two n_{elems} — length vectors, $j_i = \{j_{in} = 2b_{in} - 1 | b_{in} \sim \text{Bernoulli}(0.5), n \in [1, n_{\text{elems}}]\}$ and $k_i = \{k_{in} = 2b_{in} - 1 | b_{in} \sim \text{Bernoulli}(0.5), n \in [1, n_{\text{elems}}]\}$. Each parameter of the network is then obtained, as with the reparameterization trick, via

$$w_n(m_i) = \mu_i + \Delta w_i j_{in} k_{in}. \quad (52)$$

This can be performed very quickly using matrix multiplication, affording a decrease in the variance of the stochastic gradient by a factor of $\sim 1/n_{\text{elems}}$ in comparison to using shared parameter values for an entire minibatch for approximately twice the computational cost, although due to parallelization this can be done in equal time.

Example: Cosmological parameter inference and uncertainty calibration: The inference of cosmological parameter values from data, such as maps of the Cosmic Microwave Background radiation, is an important task in these times of precision cosmology. It is therefore useful to consider the comparison of several of the variational inference methods to calibrate their performance [35]. The study uses two CNN architectures, AlexNet and VGG to predict, from an image of the CMB, a Gaussian posterior distribution on a limited set of three cosmological parameters. The outputs of these neural networks, parameterized with w , are therefore chosen to be the mean, $\mu \equiv \mu(x, w)$, and covariance, $\Sigma \equiv \Sigma(x, w)$, of a multivariate Gaussian distribution. The loss function used to train these networks under a Flipout model is

$$\begin{aligned} \Lambda(m) = & - \mathbb{E}_{w \sim q^m} \left[\frac{1}{2} (y - \mu)^T \Sigma^{-1} (y - \mu) + \frac{1}{2} \log \det \Sigma \right] \\ & + \text{KL}(q^m \parallel \rho). \end{aligned} \quad (53)$$

In this work, the authors perform a post-training re-calibration of the models to ensure that some coverage properties are respected. In practice, they adopt the Platt Scaling method [36], to empirically adjust the posteriors as to make the reliability diagram of their coverage probability well calibrated. Note however that this simple scaling cannot account for all deviations from the true posterior shape.

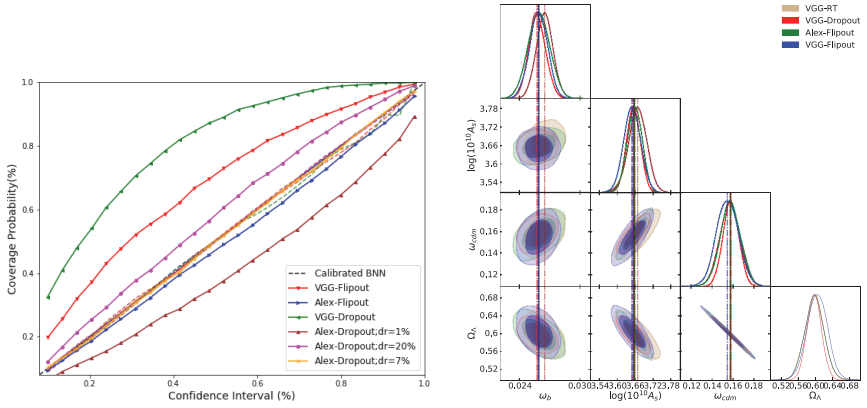


Fig. 5. Left: Reliability diagrams for different models, before (solid) and after (dashed) post-training re-calibration. Right: Approximate posteriors for cosmological parameters obtained after calibration of the models. This illustrates the difficulty of obtaining well-calibrated probabilistic models from neural networks directly out of the optimization procedure, but post-hoc calibration can correct some of these biases. Figure credit: Hortua *et al.* (2019) [35].

The results of this procedure are illustrated on Fig. 5 where the left plot shows the reliability diagrams of the various models before and after calibration. The right plot illustrates the confidence contours for the approximate posterior of cosmological parameters predicted by four different re-calibrated models on the same input data. They are fairly similar in terms of sizes, but not identical, showing that this re-calibration cannot account for complex departures in posterior shapes.

One of the takeaways of this work is that overall Flipout appear to be the best performing method in terms of calibration, training speed, and accuracy, out of the four explored (reparameterization, Flipout, MC Dropout, DropConnect).

3.2.7. *Neutra*

We can also use variational inference as part of a Markov chain sampling scheme. Neutra [37] is a method which samples from a normal distribution and then performs a bijective transformation,

$\mathbf{g} : \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I}) \rightarrow \lambda \sim \rho_\chi$, to a space approximating the target distribution. In this way, it can be seen as an approximation to the Riemannian manifold HMC (Sec. 3.1.2) where the metric is defined by the bijective function, $\mathbb{I}(\lambda) = (\mathcal{J}\mathcal{J}^T)^{-1}$, where the Jacobian is $\mathcal{J} = \partial_\epsilon \mathbf{g}$. Using neural networks (and particularly many of the modern density estimators such as inverse autoregressive flows, etc.) this Jacobian is very easy to evaluate and therefore \mathbf{g} can be arbitrarily complex, fittable to the desired function by maximizing the ELBO and quick to evaluate. Using HMC, samples can be obtained very easily from the normal distribution and the bijected forward to get samples from an approximation to the target distribution much more efficiently than samples can be obtained by directly evaluating the target distribution. It should be noted that this method, like all those mentioned in this section, is not Bayesian in nature, since, in this case, there is no quantification in how well the bijection is really performing. Therefore, there is no way to tell if the samples, $\mathbf{g}(\epsilon)$, actually coincide with samples $\lambda \sim \rho_\chi$, meaning the distribution could be very different from that desired when addressing target distribution using exact evaluations.

4. Concluding Remarks and Outlook

Despite impressive accuracy in supervised learning benchmarks, current state-of-the-art neural networks are poor at quantifying predictive uncertainty, and as such are prone to produce overconfident predictions and biases which are extremely difficult to disentangle from true properties of the data. The fact that proper uncertainty quantification is crucial for many practical applications justifies the formulation of neural networks as statistical models as a first step towards using them for inference.

While truly Bayesian neural networks have the capacity to fully characterize the epistemic uncertainty introduced by the neural network, in practice, exact Bayesian inference is intractable for neural networks. It is common to resort to either using numerically approximated by exact samples of posterior distribution of network parameters, that is, Monte Carlo methods, or to using approximate

distributions as a proxy for the true Bayesian posterior, through variational inference. The fact that, through the former method, Bayesian neural networks are often harder to train and implement than non-Bayesian neural networks means that, in the literature, variational methods have gained a lot of popularity in the recent years. However, as we have stressed in this chapter, those latter approximate methods suffer from many pitfalls, in particular the lack of guarantee that the approximate distribution is sufficiently close to the desired target.

Because of these, other statistical tools and tests should be used in concurrence with approximate Bayesian neural networks, such as calibration and test of generalization of the predictive uncertainty to domain shifts [38]. However, it is worth noting that Bayesian neural networks are not necessarily the most useful for doing the best reasoned inference of network outputs. For this, other methods, such as likelihood-free (simulation-based) inference, could be more efficient, powerful, and easier to implement.

References

- [1] K. Athreya and S. Lahiri, *Measure Theory and Probability Theory*. Springer Texts in Statistics (Springer, 2006).
- [2] P. Müller and D. Rios, Issues in bayesian analysis of neural network models, *Neural Comput.* **10** (1998) 749–70.
- [3] T. Charnock, G. Lavaux, B. D. Wandelt, S. Sarma Boruah, J. Jasche and M. J. Hudson, Neural physical engines for inferring the halo mass distribution function, *Mon. Notices Roy. Astron. Soc.* **494**(1) (2020) 50–61; doi: 10.1093/mnras/staa682.
- [4] D. Hafner, D. Tran, T. Lillicrap, A. Irpan and J. Davidson, Noise contrastive priors for functional uncertainty (2018); arXiv:1807.09289.
- [5] B. Lakshminarayanan, A. Pritzel and C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, in *Advances in Neural Information Processing Systems 30* eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, (Curran Associates, Inc., 2017), pp. 6402–6413.
- [6] E. A. J. F. Peters and G. de With, Rejection-free monte carlo sampling for general potentials, *Phys. Rev. E* **85** (2012) 026703; doi: 10.1103/PhysRevE.85.026703.
- [7] A. Bouchard-Côté, S. Vollmer and A. Doucet, The bouncy particle sampler: A non-reversible rejection-free Markov chain Monte Carlo method (2015); arXiv:1510.02451.

- [8] A. Beskos, N. S. Pillai, G. O. Roberts, J. M. Sanz-Serna and A. M. Stuart, Optimal tuning of the hybrid Monte-Carlo algorithm (2010); arXiv:1001.4460.
- [9] Y. Bouffanais and E. K. Porter, Bayesian inference for binary neutron star inspirals using a hamiltonian monte carlo algorithm, *Phys. Rev. D.* **100** (2019) 104023; doi: 10.1103/PhysRevD.100.104023.
- [10] M. D. Hoffman and A. Gelman, The no-U-turn sampler: Adaptively setting path lengths in hamiltonian Monte Carlo, *J. Mach. Learn. Res.* **15** (2014) 1351–1381.
- [11] X. Lu, V. Perrone, L. Hasenclever, Y. W. Teh and S. J. Vollmer, Relativistic Monte Carlo, in *Proc. 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017* (2017), pp. 1–11.
- [12] T. Fu and Z. Zhang, Quasi-Newton Hamiltonian Monte Carlo, in *Conf. Uncertainty in Artificial Intelligence (UAI)* (2016).
- [13] J. Jasche and B. D. Wandelt, Bayesian physical reconstruction of initial conditions from large-scale structure surveys, *Mon. Notices Roy. Astron. Soc.* **432**(2) (2013) 894–913; doi: 10.1093/mnras/stt449.
- [14] J. Jasche, F. Leclercq and B. D. Wandelt, Past and present cosmic structure in the SDSS DR7 main sample, *J. Cosmology Astroparticle Phys.* **2015**(1) (2015) 036; doi: 10.1088/1475-7516/2015/01/036.
- [15] G. Lavaux and J. Jasche, Unmasking the masked Universe: The 2M++ catalogue through Bayesian eyes, *Mon. Notices Roy. Astron. Soc.* **455**(3) (2016) 3169–3179; doi: 10.1093/mnras/stv2499.
- [16] D. Kodi Ramanah, T. Charnock and G. Lavaux, Painting halos from cosmic density fields of dark matter with physically motivated neural networks, *Phys. Rev. D* **100**(4) (2019) 043515; doi: 10.1103/PhysRevD.100.043515.
- [17] M. Girolami, B. Calderhead and S. A. Chin, Riemannian Manifold Hamiltonian Monte Carlo (2009); arXiv:0907.1100.
- [18] T. Chen, E. B. Fox and C. Guestrin, Stochastic gradient Hamiltonian Monte Carlo, in *31st Int. Conf. Machine Learning, ICML 2014*, Vol. 5, (2014), pp. 3663–3676; arXiv:1402.4102.
- [19] C. Blundell, J. Cornebise, K. Kavukcuoglu and D. Wierstra, Weight uncertainty in neural networks (2015); arXiv:1505.05424.
- [20] G. Dikov, P. van der Smagt and J. Bayer, Bayesian learning of neural network architectures (2019); arXiv:1901.04436.
- [21] D. P. Kingma, Fast gradient-based inference with continuous latent variable models in auxiliary form (2013); arXiv:1306.0733.
- [22] D. P. Kingma and M. Welling, Auto-encoding variational Bayes (2013); arXiv:1312.6114.
- [23] A. Möller and T. de Boissière, SuperNNova: An open-source framework for Bayesian, neural network-based supernova classification, *Mon. Notices Royal Astron. Soc.* **491**(3) (2020) 4277–4293; doi: 10.1093/mnras/stz3312.
- [24] M. Fortunato, C. Blundell and O. Vinyals, Bayesian recurrent neural networks (2017); arXiv:1704.02798.
- [25] M. H. DeGroot and S. E. Fienberg, The Comparison and Evaluation of Forecasters, *J. Royal Statist. Soc. Ser. D* **32**(1/2) (1983) 12–22.

- [26] D. P. Kingma, T. Salimans and M. Welling, Variational dropout and the local reparameterization trick (2015); arXiv:1506.02557.
- [27] Y. Gal and Z. Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning (2015); arXiv:1506.02142.
- [28] Y. Gal, J. Hron and A. Kendall, Concrete dropout (2017); arXiv:1705.07832.
- [29] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors (2012); arXiv:1207.0580.
- [30] R. Houthoofd, X. Chen, Y. Duan, J. Schulman, F. De Turck and P. Abbeel, VIME: Variational Information maximizing exploration (2016); arXiv:1605.09674.
- [31] L. Perreault Levasseur, Y. D. Hezaveh and R. H. Wechsler, Uncertainties in parameters estimated with neural networks: Application to strong gravitational lensing, *Astrophys. J.* **850**(1) (2017) L7.
- [32] M. Walmsley, L. Smith, C. Lintott, Y. Gal, S. Bamford, H. Dickinson, L. Fortson, S. Kruk, K. Masters, C. Scarlata, B. Simmons, R. Smethurst and D. Wright, Galaxy Zoo: Probabilistic morphology through Bayesian CNNs and active learning, *Mon. Notices Royal Astron. Soc.* **491**(2) (2020) 1554–1574; doi: 10.1093/mnras/stz2816.
- [33] D. J. C. MacKay, Information-based objective functions for active data selection, *Neural Comput.* **4**(4) (1992) 590–604; doi: 10.1162/neco.1992.4.4.590.
- [34] Y. Wen, P. Vicol, J. Ba, D. Tran and R. Grosse, Flipout: Efficient pseudo-independent weight perturbations on mini-batches, in *6th Int. Conf. Learning Representations, ICLR 2018* (2018), pp. 1–16.
- [35] H. J. Hortua, R. Volpi, D. Marinelli and L. Malagò, Parameters estimation for the cosmic microwave background with Bayesian neural networks (2019); arXiv:1911.08508.
- [36] M. Kull, T. Silva Filho and P. Flach, Beta calibration: A well-founded and easily implemented improvement on logistic calibration for binary classifiers, in *Proc. 20th Int. Conf. Artificial Intelligence and Statistics, AISTATS 2017*, Vol. 54 (2017).
- [37] M. Hoffman, P. Sountsov, J. V. Dillon, I. Langmore, D. Tran and S. Vasudevan, Neutralizing bad geometry in Hamiltonian Monte Carlo using neural transport (2019); arXiv:1903.03704.
- [38] J. Snoek, Y. Ovadia, E. Fertig, B. Lakshminarayanan, S. Nowozin, D. Sculley, J. Dillon, J. Ren and Z. Nado, Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift, in *Advances in Neural Information Processing Systems 32* eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché Buc, E. Fox and R. Garnett, (Curran Associates, Inc., 2019), pp. 13991–14002.

This page intentionally left blank

Chapter 19

Parton Distribution Functions

Stefano Forte* and Stefano Carrazza[†]

*Tif Lab, Dipartimento di Fisica, Università di Milano and
INFN, Sezione di Milano,
Via Celoria 16, I-20133 Milano, Italy
forte@mi.infn.it

[†]stefano.carrazza@mi.infn.it

We discuss the determination of the parton substructure of hadrons by casting it as a peculiar form of pattern recognition problem in which the pattern is a probability distribution, and we present the way this problem has been tackled and solved. Specifically, we review the NNPDF approach to PDF determination, which is based on the combination of a Monte Carlo approach with neural networks as basic underlying interpolators. We discuss the current NNPDF methodology, based on genetic minimization, and its validation through closure testing. We then present recent developments in which a hyperoptimized deep-learning framework for PDF determination is being developed, optimized, and tested.

1. Introduction

The determination of the parton substructure of the nucleon is essentially a pattern recognition problem: given an unknown underlying function that maps input instances to actually realized outcomes, use a set of data to infer the function itself. However, the determination of parton distributions (PDFs, henceforth) differs from standard pattern recognition problems (such as, say, face detection) in many peculiar and perhaps unique relevant aspects. Also, whereas the first PDF determinations have been performed around forty-five years ago [1–6] it was only recognized less than twenty years ago [7–11] that AI techniques could be used for PDF determination: see Fig. 1.

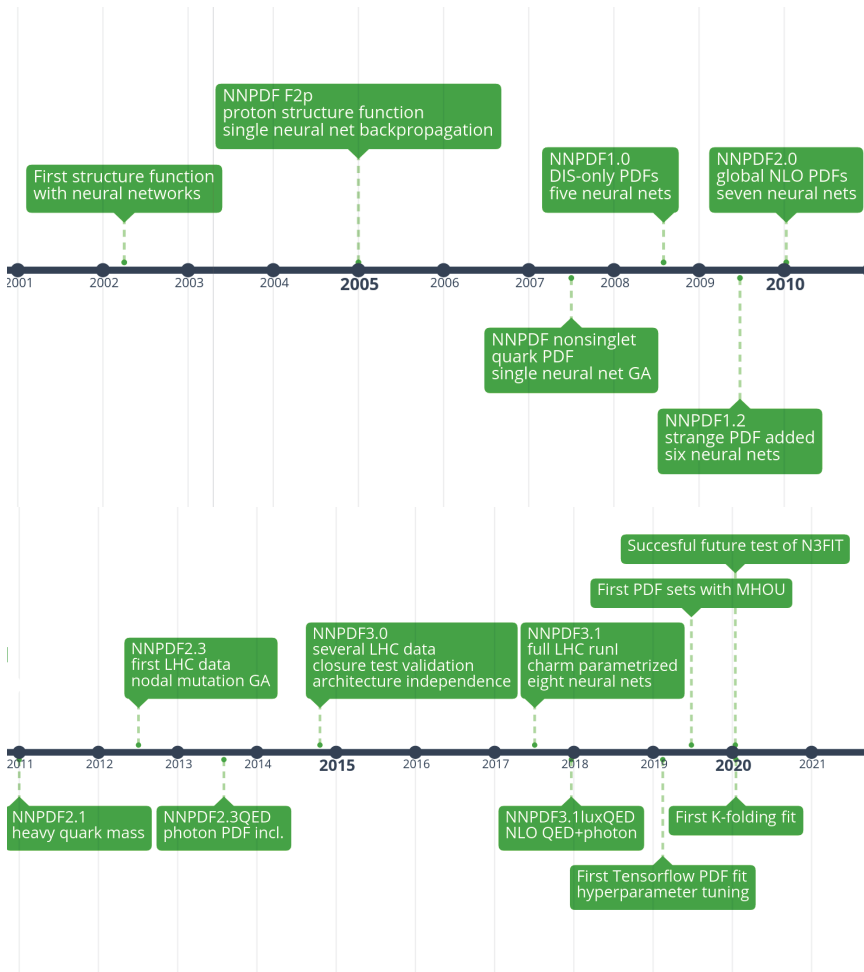


Fig. 1. Timeline for the development of PDFs based on AI techniques.

In this section, we will first briefly review what the problem of PDF determination consists of, in which sense it can be viewed as a pattern recognition problem, and the peculiarities that characterize it. We will then briefly summarize the NNPDF approach to PDF determination, which is the only approach in which the problem has been tackled using AI techniques.

In Sec. 2, we will provide a more detailed discussion of the NNPDF tool-set used for the determination of current published PDF sets, i.e. up to NNPDF3.1 [12]. We will specifically discuss the use of neural nets as PDF interpolants, PDF training using genetic minimization and cross-validation, and the validation methodology based on closure testing. In Sec. 3, we will then turn to a methodology that is currently being developed for future PDF determinations, which updates the standard AI tools used by NNPDF to more recent machine learning methods, relying on deterministic minimization, model optimization (hyper-optimization) and more powerful and detailed validation techniques.

1.1. *PDF determination as an AI problem*

PDFs encode the structure of strongly-interacting particles or nuclei, as probed in high-energy collisions. A review of the underlying theory is beyond the scope of this work, and the reader is referred to standard textbooks [13], summer school lecture notes [14] and recent specialized reviews [15, 16] for more detailed discussions. Here it will suffice to say that a generic observable, such as the total cross-section $\sigma_X(s, M_X^2)$ for a “hard” (i.e. perturbatively computable in QCD) physical process in a collision between two hadrons (such as two protons at the LHC) has the structure

$$\begin{aligned} \sigma_X(s, M_X^2) \\ = \sum_{a,b} \int_{x_{\min}}^1 dx_1 dx_2 f_{a/h_1}(x_1, M_X^2) f_{b/h_2}(x_2, M_X^2) \hat{\sigma}_{ab \rightarrow X}(x_1 x_2 s, M_X^2). \end{aligned} \quad (1)$$

Here s is the (square) center-of-mass energy of the collision (so $s = (13 \text{ TeV})^2$ at the LHC) and M_X is the mass of the final state (so $M_X = 125 \text{ GeV}$ for Higgs production); σ_X is the measurable cross-section, observed in proton–proton interactions (hadronic cross section, henceforth), while $\hat{\sigma}_{ab \rightarrow X}$ is the computable cross-section, determined in perturbation theory from the interaction of two incoming partons, i.e. quarks and gluons a and b (partonic cross-section, henceforth).

In Eq. (1) f_{a/h_1} , f_{b/h_2} are the PDFs: they provide information on the probability of extracting a parton of kind a , b (up quark, up antiquark, etc.) from incoming hadrons h_1 , h_2 . Note that PDFs are not quite probability densities, first because they are not functions but rather distributions (like the Dirac delta), and also, they are not positive definite. The PDFs are a universal property of the given hadron: e.g. the proton PDFs are the same for any process with a proton in the initial state. They depend on x , which can be viewed as the fraction of the momentum of the incoming hadron carried by the given parton, so $0 \leq x \leq 1$, and on the scale M_X^2 . The dependence on M_X^2 is computable in perturbation theory, just like the partonic cross-section $\hat{\sigma}_{ab \rightarrow X}$, and it is given as a set of integro-differential equations, having as initial conditions the set of PDFs at some reference scale Q_0 .

The dependence of the PDFs on x would be computable if one was able to solve QCD in the non-perturbative domain, i.e. if it was possible to compute the proton wave function from first principles. This is of course not the case, other than through lattice simulations [17]. Hence, in principle, PDFs for any given hadron at some reference scale Q_0 are a set of well-defined functions of x , namely $f_{a/h}(x, Q_0^2)$, which depend on the single free parameter of the theory, the strong coupling (and, for heavy quark PDFs, the heavy quark masses). We know that these functions exist, but we do not know what they are: at present, they can only be determined by comparing cross-sections of the form Eq. (1) for a wide enough set of observables for which the hadronic cross-section is measured with sufficient precision, and the partonic cross-section is known with sufficient accuracy (i.e. to high enough perturbative order in QCD, including electroweak corrections, etc.).

The traditional way the problem has been approached is by postulating a particular functional form for the x dependence of the PDFs at a reference scale Q_0 , given in terms of a set of free parameters; determining the PDFs at all other scales Q by solving perturbative evolution equations; and determining the free parameters by fitting to the data. The standard choice, adopted since the very first

attempts [1] is

$$f_i = x^{\alpha_i}(1-x)^{\beta_i}, \quad (2)$$

where now i collectively indicates the type of parton and of parent hadron. This functional form is suggested by theory arguments (or perhaps prejudice) implying that PDFs should display power-like behavior as $x \rightarrow 0$ and as $x \rightarrow 1$ (see e.g. [18]). Note that, even if this were true, there is no reason to believe that this behavior should hold for all x , and thus, given that only a finite range in x is experimentally accessible (currently roughly $10^{-4} \lesssim x \lesssim 0.5$), it is unclear that this functional form should apply at all in the observable region. Furthermore, from the equations which govern the Q^2 dependence of the PDFs, it is easy to see that even if the PDF takes the form of Eq. (2) at some scale, this form is not preserved as the scale is varied: specifically, it is corrected by $\ln x$ terms as $x \rightarrow 0$, and by $\ln(1-x)$ terms as $x \rightarrow 1$.

The fact that the simple functional form Eq. (2) is too restrictive has been rapidly recognized, and more and more elaborate functional forms have been adopted in more recent PDF determinations. For example, the gluon PDF of the proton was parameterized in the CTEQ5 [19] PDF set as

$$xg(x, Q_0^2) = A_0 x^{A_1} (1-x)^{A_2} (1 + A_3 x^{A_4}) \quad (3)$$

and in the CT18 PDF set [20] as

$$\begin{aligned} g(x, Q = Q_0) &= x^{a_1-1} (1-x)^{a_2} [a_3(1-y)^3 \\ &\quad + a_4 3y(1-y)^2 + a_5 3y^2(1-y) + y^3], \\ y &= \sqrt{x}, \quad a_5 = (3 + 2a_1)/3. \end{aligned} \quad (4)$$

Issues related to postulating a fixed functional form for PDFs were made apparent when a determination of the uncertainties on the PDFs was first attempted [21–23]. Namely, uncertainties on the fit parameters determined by least-squares and standard error propagation turned out to be smaller by about one order of magnitude than one might reasonably expect by looking at the fluctuation of best-fit

values as the underlying dataset was varied. This led to the peculiar concept of “tolerance”, namely, an *a-posteriori* rescaling factor of uncertainties. It is debatable how much of the need for such a rescaling is related to the bias introduced by the choice of a particular functional form. However, a not uncommon occurrence is that addition of new data, leading to a more extended parameterization (such as Eq. (4) in comparison to Eq. (3)) would lead to an *increase* in uncertainties. This suggests that the more restrictive parameterization might well be biased.

In 2002 it was first suggested [7] that these difficulties may be overcome by addressing the problem of PDF determination by means of a standard AI tool, neural networks. The basic underlying intuition is that neural networks provide a universal interpolating function, and that by choosing a sufficiently redundant architecture any functional form can be accommodated in a bias-free way, while avoiding overtraining through suitable training methods, as we will discuss in Secs. 2.2.2 and 3.4.1 below. This first suggestion was gradually developed into a systematic methodology for PDF determination through a series of intermediate steps (see Fig. 1) involving, on the methodological side, a number of subsequent improvements, to be discussed below, and a set of validation and testing techniques. The more recent successors NNPDF3.0 [24] and NNPDF3.1 [12] of the first PDF set developed using this methodology (NNPDF1.0 [10]) are currently the most widely cited PDF sets.

It should now be clear in which sense PDF determination can be viewed as a pattern recognition problem, and what are its peculiar features. As in standard pattern recognition, the main goal is to determine a set of unknown underlying functions from data instances, with almost no knowledge of their functional form (other than loose constraints of integrability with an appropriate measure, smoothness, etc.). Unlike in the simplest pattern recognition problems, the functions provide continuous output (i.e. the features to be recognized are continuous), and data are not directly instances of the functions to be determined. Hence, one cannot associate an input–output pair to an individual data point. Rather, as apparent from Eq. (1), each datapoint provides an output which depends in a nonlinear way on

the full set of functions evaluated at all input values, which are integrated over from some minimum x_{\min} (depending on the particular observable and the values of s and M_X^2). This is of course common to more complex pattern recognition problems, such as in computer vision.

There are however two peculiarities in PDF determination which set it apart from most or perhaps all other applications of AI. The first is that the quantities which one is trying to determine, the PDFs, are probability distributions of observables, rather than being observables themselves. This follows from the fact that, due to the quantum nature of fundamental interactions, individual events (i.e. measurement outcomes) are stochastic, not deterministic. Even if the PDF were known exactly to absolute accuracy, the cross-section would just express the probability of the observation of an event, to be determined through repeated measurements. The PDFs are accordingly probability distributions. The goal of PDF determination is to determine the probability distribution of PDFs: hence, in PDF determination one determines a probability distributions of probability distributions, i.e. a probability functional.

The second peculiarity is that in order for a PDF determination to be useful as an input to physics predictions, full knowledge of PDF correlations is needed. In fact, PDF uncertainties are typically a dominant source of uncertainty in predictions for current and future high-energy experiments [25]. But the uncertainty on each particular PDF at a given x value, $f_i(x, Q_0^2)$ is correlated to the uncertainty on any other PDF at a different x value $f_j(x', Q_0^2)$, and this correlation must be accounted for in order to reliably estimate PDF uncertainties [26]. Hence, PDF determination also requires the determination a covariance matrix of uncertainties in the space of probability distributions: namely, a covariance matrix functional.

The NNPDF approach to PDF determination tackles this problem using AI tools, as we discuss in the next section.

1.2. *The NNPDF approach*

As seen in Sec. 1.1 the NNPDF methodology has the goal of determining the probability distribution of a set of functions, which in turn

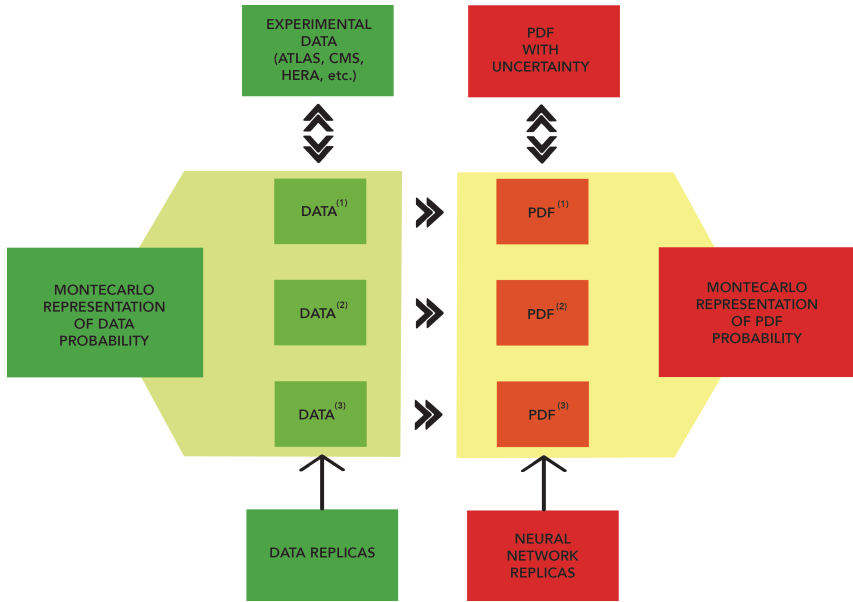


Fig. 2. Schematic representation of the NNPDF methodology.

are related to the probability distributions of quantum events (the emission of a parton from a parent hadron) that provide the input to the computation of predictions for (discrete) experimental measurements. The methodology is based on two distinct ingredients: the use of a Monte Carlo representation for the probability distributions, and the use of neural networks as unbiased underlying interpolating functions. It is schematically represented in Fig. 2.

The Monte Carlo representation provides a way of breaking down the problem of determining a probability in a space of functions into an (in principle infinite) set of problems in which a unique best-fit set of functions is determined. The basic idea is to turn the input probability distribution of data into a Monte Carlo representation. This means that the input data and correlated uncertainties are viewed as a probability distribution (typically, but not necessarily, a multigaussian) in the space of data, such that the central experimental values correspond to the mean and the correlated uncertainties correspond to the covariance of any two data. The Monte Carlo representation

is obtained by extracting a set of replica instances from this probability distribution, in such a way that, in the limit of infinite number of replicas, the mean and covariance over the replica sample reproduce the mean and covariance of the underlying distributions. In practice the number of replicas can be determined *a posteriori* by verifying that mean and covariance are reproduced to a given target accuracy.

A best-fit PDF (or rather, PDF set: i.e. one function $f_i(x, Q_0^2)$ for each distinct type of parton i) is then determined for each data replica, by minimization of a suitable figure of merit. Neural networks are used to represent the PDFs, with the value of x as input, and the value of the PDF as an output (one for each PDF). Note that the fact that the data only depend indirectly on the input functions to be determined (the PDFs) is immaterial from the point of view of the general methodology. Indeed, the problem has been reduced to that of determining the optimal PDFs for each input data replica, namely, to standard training of neural networks. However, the fact that the PDF is not trained to the data directly will have significant implications on the nature of PDF uncertainties, on their validation, and on the optimization of PDF training, as we will discuss more extensively in Secs. 2.3, 3.1 and 3.2.

The output of the process is a set of PDF replicas, one for each data replica. These provide the desired representation of the probability density in the space of PDFs. Specifically, central values, uncertainties and correlations can be computed doing statistics over the space of PDF replicas: the best-fit PDF is the mean over the set of replicas, the uncertainty on any PDF for given x can be found from the variance over the replica sample, and the correlation from the covariance.

The remaining methodological problems are how to determine the optimal neural network parameterization, how to determine the optimal PDF for each replica (i.e. the optimal neural network training) and how to validate the results. The way these issues are addressed in the current NNPDF methodology will be discussed in Sec. 2, while current work towards improving and hyperoptimizing the methodology are discussed in Sec. 3.

2. The State of the Art

The NNPDF methodology, presented in Sec. 1.2, combines a Monte Carlo approach representation of probability distributions with neural networks as basic interpolants. Here we discuss first, the architecture of the neural networks, then their training, which is achieved by combining genetic minimization with stopping based on cross-validation, and finally the validation of results through closure testing.

2.1. Neural networks for PDFs

In all NNPDF determinations, starting with the proof-of-concept determination of a single PDF (isotriplet combination) in [9], up to and including the most recent global PDF set, NNPDF3.1 [12] the PDF architecture has been unchanged. Namely, PDFs are parameterized at a reference scale Q_0 and expressed in terms of a set of independent neural networks multiplied by a preprocessing factor. Each of these neural networks consists of a fixed-size feed-forward multi-layer perceptron with architecture 2-5-3-1 (see Fig. 3). The only change in subsequent releases is in the number of independently parameterized PDFs (or PDF combinations), and thus of

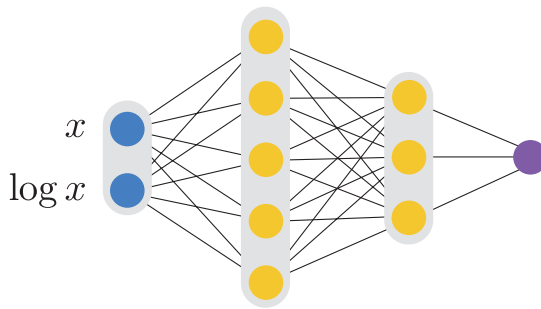


Fig. 3. Architecture of the neural networks used for PDF parameterization in all available NNPDF sets. Each PDF is parameterized by a preprocessed neural network, according to Eq. (5). The values of x and $\ln x$ are taken as input, and the value of the PDF is given as output. The number of independently parameterized PDFs has increased over time but the architecture has remained the same.

independent neural networks: one in the proof-of-concept [9], five in NNPDF1.0 [10] (up and down quarks and antiquarks and the gluon), seven from NNPDF1.1 [27], eight in NNPDF3.1 [12] (up, down, strange quarks and antiquarks; total charm; gluon).

The PDF momentum fraction x enters the input layer nodes as $(x, \log(x))$, in order to account for the fact that the physical behavior of PDFs typically has two different regimes in the physically accessible $10^{-4} \lesssim x \lesssim 0.5$ region: a linear regime in the region $0.03 \lesssim x \lesssim 0.5$ and a logarithmic regime in the region $10^{-4} \lesssim x \lesssim 0.03$. The next two hidden layers, with 5 and 3 nodes respectively, use the sigmoid activation function while the output node is linear. This particular choice of architecture was originally selected through systematic manual scans, as being sufficiently redundant to accommodate the PDF shape in an unbiased way .

The fact that it was never necessary to update this initial choice has validated the robustness of this analysis. Furthermore, in [10], it was explicitly checked that results would be unchanged if the number of nodes in the first hidden layer was reduced from 5 to 4. In [24], within a closure test (see Sec. 2.3 below), it was checked that results were unchanged if the number of the nodes in the intermediate layers was increased respectively from 5 to 20 and from 3 to 15, which corresponds to an increase of the number of free parameters of the neural net by more than one order of magnitude.

The parameterization for each PDF (or independent combination of PDFs) is

$$xf_i(x, Q_0) = A_i x^{-\alpha_i+1} (1-x)^{\beta_i} \text{NN}_i(x), \quad (5)$$

where NN_i is the neural network corresponding to a given combination i . The quantities which are independently parameterized are the linear combination of light quark and gluon PDFs which correspond to eigenvectors of the PDF Q^2 evolution equations, and charm: $\{g, \Sigma, V, V_3, V_8, T_3, T_8, c^+\}$ (see [12, 14] for the precise definition). A_i is an overall normalization constant which enforces sum rules (such as the fact that the total momentum fractions carried by all partons must add up to one) and $x^{-\alpha_i}(1-x)^{\beta_i}$ is a preprocessing factor which controls the PDF behavior at small and large x .

The preprocessing exponents α_i and β_i were initially (NNPDF1.0 [10]) chosen to be fixed, while checking that no strong dependence of results was observed upon their variation. As the accuracy of the PDF determination improved, starting with NNPDF1.2 [28], in order to ensure unbiased results, the exponents were varied. Namely, the values of α_i , β_i were randomly selected for each PDF in each replica, with uniform distribution within a range fixed for each PDF, and kept fixed during the minimization of the replica. Effectively, with reference to Fig. 2, this means that for each PDF replicas the PDF parameterization is different, because the preprocessing function of each PDF is different. The range for each type of PDF (gluon, up quark, etc) was initially determined by requiring stability of the fit results, which, starting with NNPDF2.0 [11] was quantitatively determined by computing the correlation coefficient between the figure of merit χ^2 (see Eq. (6) below) and verifying that it remained small. Starting with NNPDF3.0 [24], the range is now determined self-consistently: the effective exponents are computed for each independent combination of PDFs and for each PDF replica, the 68% confidence level range is determined for each combination, the fit is repeated with the exponents varied in a range taken equal to twice this range, and the procedure is iterated until the range stops changing.

As already mentioned, unlike in many standard regression problems, in which during the optimization procedure the model is compared directly to the training input data, in PDF fits the data are compared to theoretical predictions for physical observables of the form of Eq. (1), in which the PDFs $f_i(x, Q^2)$ are in turn obtained by solving a set of integro-differential equations from the PDFs $f_i(x, Q_0)$, parameterized at the initial scale. Hence, the observable depends on the PDF through a number of convolution integrals, between the PDFs at scale Q_0 , the evolution factors that take them to scale Q and the partonic cross-sections of Eq. (1). In practice, the convolutions are turned into multiplication of pre-computed tables (FastKernel or FK-tables) by projecting on suitable basis functions, as discussed in [11, 29], see also Sec. 3.1 below.

2.2. The minimization procedure

The optimization procedure implemented in NNPDF consists in minimizing the loss function

$$\chi^2 = \sum_{i,j}^{N_{\text{dat}}} (D - P)_i \sigma_{ij}^{-1} (D - P)_j, \quad (6)$$

where D_i is the i th data point, P_i is the convolution product between the FastKernel tables for point i and the PDF model, and σ_{ij} is the covariance matrix between data points i and j . The covariance matrix includes both uncorrelated and correlated experimental statistical and systematic uncertainties, as given by the experimental collaborations. Multiplicative uncertainties (such as normalization uncertainties), for which the uncertainty is proportional to the observable, must be handled through a dedicated method in order to avoid fitting bias: the t_0 method has been developed [30] to this purpose, and adopted from NNPDF2.0 [11] onward. Theory uncertainties (such as missing higher order uncertainties) could also be included as discussed in [31, 32] but this has only been done in preliminary PDF sets so far. Once again, we stress that input data are not provided for the neural networks, but rather for a complicated functional of the neural network output.

2.2.1. Genetic minimization

The minimization implemented in NNPDF3.1 and earlier releases is based on genetic algorithms (GA). Given that each PDF replica is completely independent from each other, the minimization procedure can be trivially parallelized. Genetic minimization was chosen for a number of reason. On the one hand, it was felt that that a deterministic minimization might run the risk of ending up in a local minimum related to the specific network architecture. Also, no efficient way of determining the derivative of the observables with respect to the parameters of the neural network was available then. In fact, modern, efficient deterministic minimization methods [33, 34] were not

yet available at the time. As we will discuss in Sec. 3.1 below, these motivations are no longer valid and deterministic minimization is now more desirable.

The GA algorithm consists of three main steps: mutation, evaluation and selection. These steps are performed subsequently through a fixed number of iterations. The procedure starts with the initialization of the neural network weights for each PDF flavor using a random Gaussian distribution. From this initial network, a number of copies is produced, for which the weights are then mutated with a suitable rule. The mutations with lowest values of the figure of merit are selected and the procedure is iterated.

The GA initially adopted was based on point change mutations, in which individual weights or thresholds in the networks were mutated at random, according to a rule of the form

$$w_i \rightarrow w_i + \eta_i r_i, \quad (7)$$

where w_i is the i th neural network weight or threshold, η_i is a mutation rate size, r_i is a uniform random number within $[-1, 1]$. A fixed number of randomly chosen parameters are then mutated for each PDF, thereby producing a given number of mutants for each generation. The GA is fully specified by assigning: (i) the number of mutations for each PDFs; (ii) the mutation rates for each mutation and for each PDF; (iii) the number of mutants for each generation; (iv) the maximum number of generations. The mutation rates were dynamically adjusted as a function of the number of iterations according to

$$\eta_i = \frac{\eta_i^{(0)}}{N_{\text{ite}}^p}. \quad (8)$$

Several subsequent versions of this GA have been adopted. In a first version (NNPDF1.0 [10]), a fixed value of the number of mutations (two per PDF), of the number of mutants ($N_{\text{mut}} = 120$) and of the exponent p ($p = 1/3$) of Eq. (8) were adopted, with a small maximum number of generations ($N_{\text{max}} = 5000$). At a later stage (NNPDF2.0 [11]) the minimization was divided in two epochs,

with a transition at $N_{\text{ite}} = 2500$ generations, and a larger number ($N_{\text{mut}} = 80$) of mutants in the first epoch, substantially decreased ($N_{\text{mut}} = 10$) in the second epoch; also, the exponent p was now randomly varied between 0 and 1 at each generation and the maximum number of generations was greatly increased ($N_{\text{max}} = 30000$). At a yet later stage (NNPDF2.3 [35]) the number of mutations was increased to three for several PDFs.

Subsequent versions of the GA also involved various reweighting procedures, in which the contribution of different datasets to the figure of merit Eq. (6) was assigned a varying weight during the training, in order to speed up the training in the early stages. In a first implementation [10], these weights were computed as a ratio of the χ^2 per datapoint for the given dataset, compared to the χ^2 per datapoint of the worst-fitted datasets, so that best-fitted dataset would get less weight. Weights were then switched off when the value of the figure of merit fell below a given threshold. In a subsequent implementation [11], the weights were computed as ratios of the χ^2 to a target χ^2 value for the given dataset (determined from a previous fit) and only assigned to datasets for which the fit quality was worse than the target. Weights were only applied in a first training epoch.

Starting with NNPDF3.0 [24], a GA based on nodal mutation has been adopted. In nodal mutation, each node in each network is assigned an independent probability of being mutated. If a node is selected, its threshold and all of the weights are mutated according to Eqs. (7) and (8), with now η fixed, and p a random number between 0 and 1 shared by all of the weights. The values $\eta = 15$ and mutation probability 15% per node have been selected as optimal based on closure tests (see Sec. 2.3 below). This algorithm proved to be significantly more efficient (see Fig. 4 below) than the previous point mutation: in particular, reweighting was no longer necessary and it was no longer necessary to have different training epochs.

2.2.2. Stopping criterion

The GA presented in Sec. 2.2 can lead to overfitting, in which not only the underlying law is fitted, but also statistical noise which

is superposed to it. In order to avoid this, a stopping criterion is required. This was implemented since NNPDF1.0 through cross-validation. Namely, the data are separated in a training set, which is fitted, and a validation set, which is not fitted. The GA minimizes the χ^2 of the training set, while the χ^2 of the validation set is monitored along the minimization, and the optimal fit is achieved when the validation χ^2 stops improving. This means that the fit optimizes the validation χ^2 , which is not fitted. Because statistical noise is uncorrelated between the training and validation sets, this guarantees that overfitting of the statistical noise is avoided. Note that more subtle forms of overfitting are possible, due to remaining correlations between training and validation sets: this, and the way to avoid it, will be discussed in Sec. 3.3 below.

In PDF fits before NNPDF3.0 [24] this stopping criterion was implemented by monitoring a moving average of the training and validation χ^2 , and stopping when the validation moving average increased while the training moving average decreased by an amount which exceeded suitably chosen threshold values. This was necessary in order to avoid stopping on a local fluctuation, and it required the tuning of the moving average and of the threshold values, which was done by studying the typical fluctuations of the figure of merit. This clearly introduced a certain arbitrariness.

Since NNPDF3.0 [24], the previous stopping criterion has been replaced by the so-called *look-back* method. In this method, the PDF parameterization is stored for the iteration where the fit reaches the absolute minimum of the validation χ^2 within a given maximum number of generations. This guarantees that the absolute minimum of the validation χ^2 within the given maximum number of iterations is achieved. The method reduces the level of arbitrariness introduced in the previous strategy, but it requires reaching the maximum number of iterations for all replicas, out of which the absolute minimum is determined. This maximum must be chosen to be large enough that the absolute minimum is always reached, and it therefore leads on average to longer training. Adoption of this new stopping has been made possible thanks to greater computing efficiency.

2.3. Closure tests

As mentioned in Sec. 1 a critical issue in PDF determination is making sure that PDF uncertainties are faithful. Therefore, the validation of a PDF set chiefly consists of verifying that PDF uncertainties accurately reproduce the knowledge of the underlying true PDFs which has been learnt and stored, together with its uncertainty, in the Monte Carlo replica set through the training procedure. Because the true PDFs are not known, this can only be done through closure testing [36]. Namely, a particular underlying truth is assumed (in our case: a specific form for the true underlying PDFs); data are then generated based on this underlying truth; the methodology is applied to this data; results are finally compared to the underlying truth.

This exercise was performed for the NNPDF3.0 PDF set [24]; since the subsequent NNPDF3.1 PDF set [12] is based on the same methodology, this provides a validation of the current NNPDF PDF sets. In this section, we will briefly review the closure testing methodology and results of [24], while the ongoing validation of the new methodology of Sec. 3 will be discussed in Sec. 3.4 below.

In this closure test, data were generated by assuming that the underlying PDF has the form of the MSTW08 PDF set [37], and then generating a dataset identical to that used for the NNPDF3.0 PDF determination (about 4000 data points) but computing the hadronic cross-sections using Eq. (1) with these PDFs adopted as input and the partonic cross-sections determined using NLO QCD theory. Clearly, the exact form of the theory is immaterial if the same theory is used to generate the data and then to fit them, in such a way that only the fitting methodology is being tested. The independence of result on the particular choice of underlying truth can be explicitly tested by repeating the procedure with a different choice for the underlying PDF.

Besides providing a validation of the NNPDF methodology, the closure test also allows for an investigation of the sources of PDF uncertainty in a controlled setting. To this purpose, three sets of closure testing data were generated in [24]. The first set (“level 0”) consists of data generated with no uncertainties. This would correspond

to a hypothetical case in which there are no experimental statistical or systematic uncertainties, so all data correspond to the “truth”, with vanishing uncertainty. A second set of data (“level 1”) is generated by assuming the probability distribution which corresponds to the published experimental covariance matrix. These data correspond to a hypothetical set of experimental results for which the experimental covariance matrix is exactly correct. A final set of data (“level 2”) is generated by taking the level 1 data as if they were actual experimental data, and then applying to them the standard NNPDF methodology, which, as discussed in Sec. 1.2 (see Fig. 2) is based on producing a set of Monte Carlo replicas of the experimental data: the level 2 data are then the Monte Carlo replicas produced out of the level 1 data, as if the latter were actual experimental data.

A first very simple test consists of fitting level 0 data, and computing the figure of merit (χ^2 per datapoint) as the training proceeds. Because these data have no uncertainty, a perfect fit with χ^2 is in principle possible. Results are shown in Fig. 4 for the two implementations of the minimization algorithm adopted in [35] (NNPDF2.3)

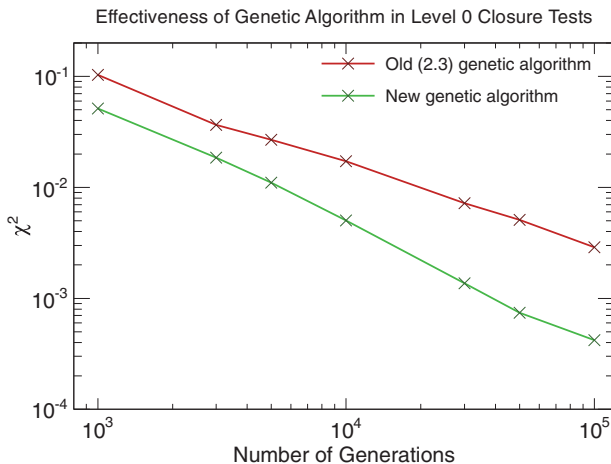


Fig. 4. The normalized figure of merit computed for the average over PDF replicas vs. the number of generations of the genetic algorithm for two different GA implementations, in a test case in which the figure of merit vanishes asymptotically.

and [24] (NNPDF3.0) and discussed in Sec. 2.2. Two sets of conclusions may be drawn from his plot. First, it is clear that the methodology is general and powerful enough to reproduce the underlying data: the figure of merit can be made arbitrarily small, which means that with vanishing experimental uncertainties, the data can be fitted with arbitrarily high accuracy. Second, it is possible to determine the dependence of the figure of merit on the training length, and specifically compare different minimization algorithms. Interestingly, Fig. 4 shows that for the two GAs of Sec. 2.2 the figure of merit follows a power law: $\chi^2 \sim \frac{1}{N\lambda}$. Furthermore, it is clear that the value of λ is rather larger (faster convergence) for the NNPDF3.0 GA, based on nodal mutation (recall Sec. 2.2), in comparison to the previous NNPDF2.3 GA implementation.

A second test compares the uncertainty on PDFs which is found when fitting respectively to level 0, level 1 and level 2 data. Results are shown for the gluon in Fig. 5: 68% confidence levels are shown for fits to level 0, level 1 and level 2 data. The plot has various implications. The first observation is that, as discussed in Sec. 1 the data constrain the PDFs only in a limited $10^{-2} \lesssim x \lesssim 0.5$ range (“data region”, henceforth). Outside that range the uncertainty grows very large, and in the absence of experimental information it is essentially arbitrary.

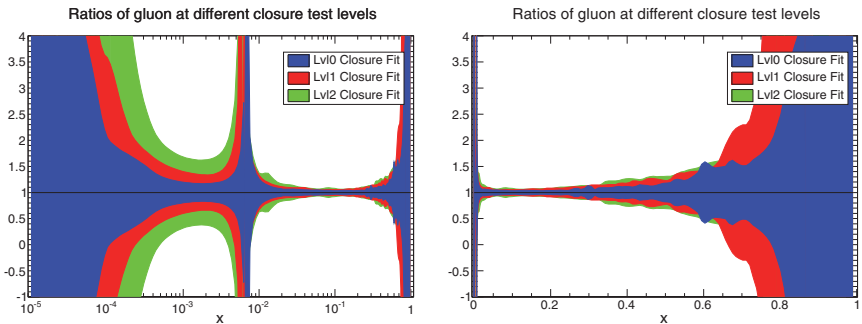


Fig. 5. The 68% confidence level uncertainty bands for the gluon PDF determined using level 0, level 1 and level 2 closure test data (see text). Results are shown vs. x at the PDF parameterization scale on a logarithmic (left) and linear (right) scale.

Coming now to the region where the experimental information is concentrated, note that when fitting level 0 and level 1 data the same datapoints are fitted over and over again, yet a spread of results is found. In the case of level 0 data we know from Fig. 4 that the figure of merit on datapoints essentially vanishes (i.e. the fit goes through all datapoints with zero uncertainty). This then means that this unique minimum at the level of data does not correspond to a unique minimum at the level of PDFs: the datapoints are measurements of the hadronic cross-section σ (see Eq. (1)), which only indirectly depends on the PDFs f_i . There is then a population of PDFs which lead to the same optimal fit because of the need to effectively interpolate between datapoints (“interpolation uncertainty”). Namely, even though at the data level there is a unique best fit, this does not correspond to a unique best-fit set of underlying PDFs.

At level 1 the datapoints are fluctuated about their true values, so the best-fit value of figure of merit on datapoints is now of order of $\chi^2 \sim 1$ per datapoint. The uncertainty is correspondingly increased because now there may be several PDF configurations which all lead to values of the figure of merit of the same order, possibly corresponding to different underlying functional forms for the PDFs (“functional uncertainty”). In other words, now the prediction is no longer uniquely determined even at the data level. Finally, at level 2, corresponding to a realistic situation, the data themselves fluctuate about the true value thereby inducing a “data uncertainty” on the PDFs.

Figure 4 shows that for the gluon in the data region these three components of the uncertainty are roughly of similar size. Note that, if a fixed functional form was fitted to the data by least-squares, both the level 0 and level 1 uncertainties would necessarily vanish. Hence, to the extent that the final level 2 uncertainty is faithful, a methodology based on a fixed functional form, for which level 0 and level 1 uncertainties vanish, necessarily leads to uncertainty underestimation.

This begs the question of checking whether indeed the level 2 uncertainties, namely, the uncertainties found with standard NNPDF methodology are faithful. A first qualitative check can be done by simply comparing the final result to the underlying truth, which in a

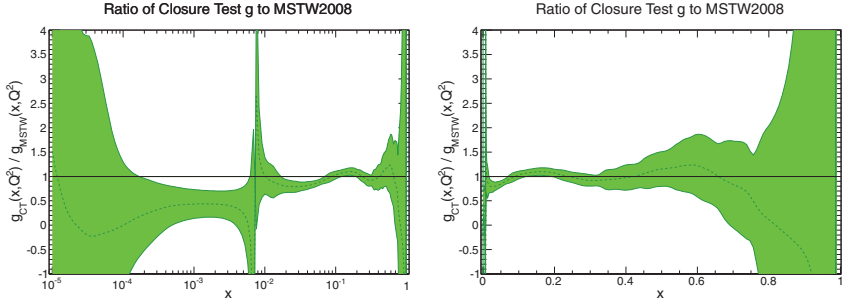


Fig. 6. The best fit gluon compared to the underlying truth, shown vs. x at the PDF parameterization scale on a logarithmic (left) and linear (right) scale. The green band is the one- σ uncertainty and the result is shown as a ratio to the underlying truth.

closure test is known. This is done for the gluon in Fig. 6. It is clear that the result appears to be broadly consistent: the truth is mostly within the one- σ band, though not always, which is as it should be, given that the one- σ band is supposed to be a 68% confidence level. Note, however, that PDF values at neighboring points in x are highly correlated: this is already true at the level of single replicas, but even more for the final PDF, obtained averaging over replicas, and it is of course as it should be — after all, if we were able to compute the PDF from first principles, it would be given by a unique functional form, most likely infinitely differentiable in the $0 < x < 1$ physical range. Hence, a confidence level cannot be computed by simply counting how many point in x space fall within the one- σ band.

Rather, a quantitative check that the confidence level is correctly determined requires repeating the whole procedure several times. Namely, we need to check that if we regenerate a set of (level 1) experimental values, and then refit them, in 68% of cases for each PDF at each point $f_i(x)$ the true value falls within the one- σ uncertainty. More in general, the validation of the PDF determination requires first, computing PDFs and uncertainties from a given set of level 2 data, so the PDF and uncertainty are obtained by taking mean and covariance over replicas. Next, repeating the determination for different sets of level 2 data obtained from different primary level 1 data: for each fit one will obtain a different best-fit PDF set

and corresponding uncertainties. Finally, computing the distribution of best-fit PDFs about the true value, and comparing this actual distribution of results about the truth with their nominal uncertainty.

In practice, the procedure is quite costly as it requires producing a large enough number of fits that confidence levels can be reliably computed, each containing a large enough set of PDF replicas that the PDF uncertainty can be reliably determined: for example, 100 sets of 100 PDF replicas each. In [24], this was done by introducing two approximations. First, the distribution of averages of level 2 replicas, each from a different set of level 1 data, was approximated with the distribution of fits of a single replica to unfluctuated level 1 data. Second, the uncertainty was assumed to be stable between different fits and was thus determined from a single 100-replica set to a particular set of level 2 data. The validity of these approximations will be further discussed in Sec. 3.4.2 below.

This procedure was used in [24] to compute the deviation of best-fit PDFs from the truth for all fitted PDFs evaluated at three x values: $x = 0.05$, $x = 0.1$ and $x = 0.2$, and respective uncertainties. The histogram of normalized deviations is compared to a univariate Gaussian in Fig. 7. The deviation between the predicted and observed

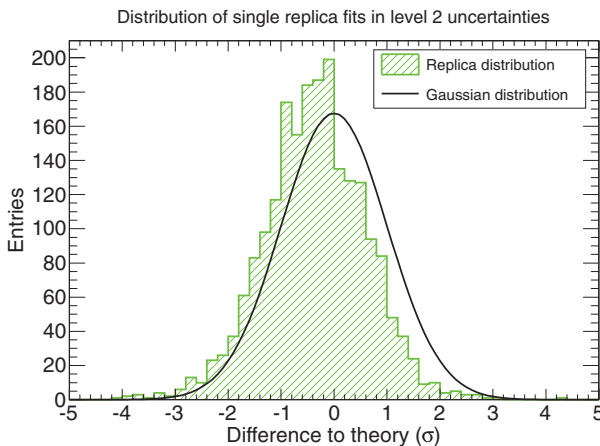


Fig. 7. Distribution of deviation between the PDF and the underlying truth normalized to its nominal uncertainty, compared to a univariate Gaussian. Results are obtained sampling all fitted PDFs at three points in x .

probability distribution are small: for instance, the one- σ confidence level is 69.9%, to be compared to the expected 68.3%. It is clear that the validation is successful.

The availability of closure test data allows performing a variety of further tests, all of which were done in Ref. [24]. On the one hand, it is possible to compare to the truth various features of the distribution of fitted PDFs, such as for example their arc-lengths, or the behavior of their probability distribution upon updating via Bayes' theorem. On the other hand, it is possible to test the stability of results upon a number of variations of the methodology, such as the choice of architecture of the neural nets, the choice of GA and its parameters, the choice of PDF parameterization basis, the parameters of the cross-validation. Indeed, as mentioned in Sec. 2.1 it has been possible to check stability upon enlarging the architecture of the neural net, as mentioned in Sec. 2.2 the method was used in order to optimize the parameters of the GA, and as mentioned above, it has been used to check the stability with respect to different choices of underlying truth.

3. The Future of PDFs in a Deep Learning Framework

The AI-based approach to PDF determination described in Sec. 2 largely eliminates potential sources of bias, specifically those related to the choice of a functional form, as discussed in Sec. 1.1, thanks to the universal nature of neural networks [38]. However, neural networks themselves are not unique, and the algorithms used for their training even less so. The methodology discussed in Sec. 2 has been developed over the years through a long series of improvements, as described in Secs. 2.1 and 2.2. These were based on trial and error, and on the experience accumulated in solving a problem of increasing complexity. The human intervention involved in these choices might in turn be a source of bias. A way of checking whether this is the case, and then improving on the current methodology, is through hyperoptimization, namely, automatic optimization of the methodology itself. This goal was recently accomplished, but it required as a prerequisite a redesign of the NNPDF codebase, and specifically the replacement

of the GA with deterministic minimization. Here we will discuss first, this code redesign, next the hyperoptimization procedure, then quality control, which plays a role analogous to cross-validation but now at the hyperoptimization level, and finally, the set of validation tests that ensure the reliability of the final hyperoptimized methodology.

3.1. A new approach based on deterministic minimization

The NNPDF methodology presented in Sec. 2 was implemented by the NNPDF collaboration as an in-house software framework relying on few external libraries. There are two major drawbacks of such an approach. First, the in-house implementation greatly complicates the study of novel architectures and the introduction of the modern machine learning techniques developed during the last decade. Second, the computational performance of GA minimization algorithms is a significant limitation, and it drastically reduces the possibility of performing hyperparameter scans systematically.

In order to overcome these problems the code has been redesigned using an object-oriented approach that provides the required functionality to modify and study each aspect of the methodology separately, and a regression model has been implemented from scratch in a modular object-oriented approach based on external libraries. Keras [39] and TensorFlow [40] have been chosen as back-ends for neural network and optimization algorithms. This code design provides an abstract interface for the implementation of other machine learning oriented technologies, that simplifies maintainability and opens the possibility to new physics studies.

The new framework implements gradient descent (GD) methods to replace the previously used GA described in Sec. 2.2. Thanks to state-of-the art tools, this change reduces the computing cost of a fit while achieving similar or better goodness-of-fit. The GD methods produce more stable fits than their GA counterparts, and, thanks to the back-ends, the computation of the gradient of the loss function is efficient even when including the convolution with the FastKernel tables discussed in Sec. 2.1. Given the possibility of performing

hyperoptimization scans, there is no longer a risk of ending up in architecture-dependent local minima.

In terms of neural networks, the new code uses just one single densely connected network as opposed to a separate network for each flavor. As previously done, we fix the first layer to split the input x into the pair $(x, \log(x))$. We also fix eight output nodes (one per flavor) with linear activation functions. Connecting all different PDFs we can directly study cross-correlation between the different PDFs not captured by the previous methodology.

As we change both the optimizer and the architecture of the network, the optimal setup must be re-tuned from scratch. To this purpose, we have implemented the hyperopt library [41], which allow us to systematically scan over many different combinations of hyperparameters finding the optimal configuration for the neural network. Therefore, the neural network architecture no longer has the form shown in Fig. 3: first, rather than a neural net per PDF, there is now a single neural net with as many outputs as are the independent PDFs, and second, the architecture (number of intermediate layers and number of nodes per layer) is now hyperoptimized, rather than being fixed.

In Fig. 8, we show a graphical representation of the full new methodology which will be referred to as **n3fit** in the sequel. The $x\text{grid}_1, \dots, x\text{grid}_n$ are vectors containing the x -inputs of the neural

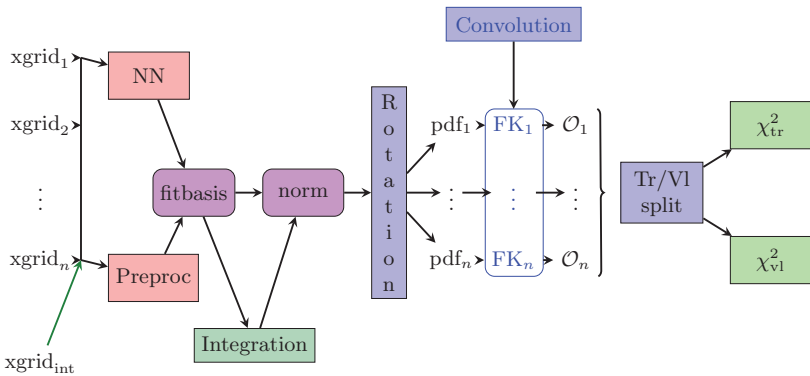


Fig. 8. Diagrammatic view of the **n3fit** code (from [42]).

network for each of the datasets entering the fit. These values of x are used to compute both the value of the neural network and the preprocessing factor, thus determining the unnormalized PDF. The normalization constants A_i (see Eq. (5)) are computed at every step of the fitting using the $x_{\text{grid}_{\text{int}}}$ points. Recall from Sec. 2.1 that the PDFs are parameterized in a basis of linear combinations $\{g, \Sigma, V, V_3, V_8, T_3, T_8, c^+\}$: individual PDFs for the quark flavors, antiflavors and the gluon, $\{\bar{s}, \bar{u}, \bar{d}, g, d, u, s, c(\bar{c})\}$, are obtained through a rotation. This procedure concludes the necessary operations to compute the value of the PDF for any flavor at the reference scale Q_0 .

All PDF parameters are stored in two blocks, the first named NN, namely the neural network of Eq. (5), and the preprocessing α and β . Given that each block is completely independent, we can swap them at any point, allowing us to study how the different choices affect the quality of the fit. All the hyperparameters of the framework are also abstracted and exposed. This specifically allows us to study several architectures hitherto unexplored in the context of PDF determination.

As repeatedly discussed in Secs. 1 and 2, the PDFs are not compared directly to the data, but rather, predictions are obtained through a convolution over the neural networks. This, as mentioned in Sec. 2.1, is performed through the FastKernel method, which produces a set of observables $\mathcal{O}_1 \dots \mathcal{O}_n$ from which the χ^2 Eq. (6) can be computed. For this purpose, the first step is generation of a rank-4 luminosity tensor

$$\mathcal{L}_{i\alpha j\beta} = f_{i\alpha} f_{j\beta}, \quad (9)$$

where (i, j) are flavor indices while (α, β) label the index on the respective x grids. Typical grids have of order of a hundred points in x for each PDF, spaced linearly in x at large $x > 0.1$, and logarithmically at small x ; the grids are benchmarked and optimized in order to guarantee better than percent accuracy with high computational efficiency [11, 12, 29]. The physical observable, e.g. an inclusive cross-section or differential distribution, is then computed by contracting the luminosity tensor with the rank-5 FastKernel table

for each separate dataset,

$$\mathcal{O}_n = \text{FK}_{i\alpha j\beta}^n \mathcal{L}_{i\alpha j\beta}, \quad (10)$$

where n corresponds to the index of the experimental data point within the dataset. This stage of the model is the most computationally intensive.

As discussed in Sec. 2.2.2, the optimal fit is determined through cross-validation. The cross-validation split, which takes the output and creates a mask for the training and validation sets, is introduced as a final layer. As mentioned, the training set is used for updating the parameters of the network during the fit while the validation set is monitored during the fit and only used for early stopping purposes. In Fig. 9, we present a schematic view of the stopping algorithm implemented in `n3fit`. The training is performed until the validation stops improving, from that point onward we enable a patience algorithm which waits for a number of iterations before raising the stopping action. For post-processing purposes we only accept stopping points for which the PDF produces positive predictions for a subset of pseudo data which tests the predictions for multiple processes in different kinematic ranges, see [12, 24] for further details.

The loss function Eq. (6) is minimized using gradient descent. Faster convergence and stability are found using algorithms with

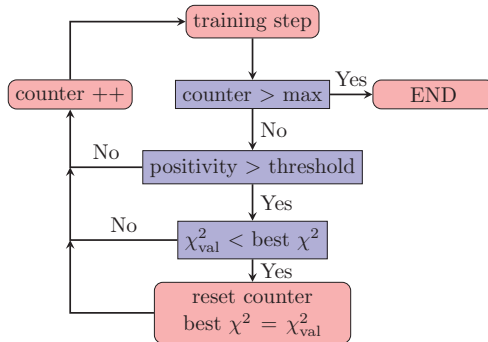


Fig. 9. Flowchart describing the patience algorithm of the `n3fit` code (from [42]).

adaptive moment, in which the learning rate of the weights is dynamically modified, such as Adadelta [33], Adam [34] and RMSprop [43]. These three optimizers adopt similar gradient descent strategies, but differ in the prescription for weight update.

This approach has been applied to the baseline setup of the NNPDF3.1 NNLO PDF determination [12]: specifically, adopting the same dataset and cuts, together with the same fraction of validation data for cross-validation, though now the stopping criterion is different (Fig. 9). This setup, henceforth referred to as “global”, includes all datasets used in NNPDF3.1 NNLO, with 4285 data points. We also studied a reduced dataset which only includes data from deep-inelastic scattering (DIS), which is computationally less intensive, in particular because DIS is an electroproduction process, so the integral in Eq. (1) only involves a single PDF. This setup, called “DIS”, includes 3092 data points, and it facilitates the process of benchmarking and validation, since it leads to computationally very light fits, which allow us to extensively explore the parameter space.

In summary, the new methodology considerably improves the computational efficiency of PDF minimization, in particular because GD methods improve the stability of the fits, producing fewer bad replicas which need to be discarded, than their GA counterparts. This translates in a much smaller computing time. The old and new algorithms are compared in Table 1: we find a factor of 20 improvement with respect to the old methodology and near to a factor of 1.5 in the percentage of accepted replicas for a global fit setup. In terms of memory, in the old methodology usage is driven by the APFEL [44]

Table 1. Comparison of the average computing resources consumed by the old and new methodologies for the DIS and Global setups.

DIS fit	CPU h.	Mem. Usage (GB)	Good replicas
n3fit (new)	0.2	2	95%
nnfit (old)	4	4	70%
Global fit	CPU h.	Mem. Usage (GB)	Good replicas
n3fit (new)	1.5	4	95%
nnfit (old)	30	5	70%

code used in order to solve PDF evolution equations, which does not depend on the set of experiments being used. In the new code, evolution is never called during the fit (it is pre-computed in the `fktables` and then the final PDFs are evolved to all scales offline), so memory consumption is driven by the TensorFlow optimization strategy which in the case of hadronic data requires the implementation of Eq. (10) and its gradient. This difference translates to an important decrease on the memory usage of `n3fit`.

3.2. *Optimized model selection*

The main motivation for the development of the new optimized code discussed in Sec. 3.1 is the possibility of performing systematic explorations of the methodology through hyperoptimization. Firstly, the new design of the `n3fit` code exposes all parameters of the fit including the neural network architecture. This is of key importance for a proper hyperparameter scan where everything is potentially interconnected. Furthermore, the new methodology has such a smaller impact on computing resources that many more fits can be performed, with a difference by several orders of magnitude: for each fit using the old methodology hundreds of setups can now be tested.

The hyperparameter scan procedure has been implemented through the `hyperopt` framework [41], which systematically scans over a selection of parameter using Bayesian optimization [45], and measures model performance to select the best architecture. Table 2 displays an example of selection of scan parameters, subdivided into those which determine the Neural Network architecture, and those which control the minimization.

Hyperparameter scans have been performed both in global and DIS setups. The best model configuration has been searched for, using as input data the original experimental values, rather than the data replicas which are then used for PDF determination (recall Sec. 1.2). Optimization has been performed using a combination of the best validation χ^2 and stability of the fits: specifically, the architecture which produces the lowest validation χ^2 has been selected after having trimmed combinations which displayed unstable behavior.

Table 2. Parameters on which the hyperparameter scan is performed from [42].

Neural network	Fit options
Number of layers	Optimizer
Size of each layer	Initial learning rate
Dropout	Maximum number of epochs
Activation functions	Stopping Patience
Initialization functions	Positivity multiplier

An example of scan for some of the parameters shown in Table 2, based the DIS setup, is shown in Fig. 10. The results of this scan can be summarized as follows. The Adadelta optimizer, for which no learning rate is used, is found to be more stable, and to systematically produce better results than RMSprop and Adam with a wide choice of learning rates. The initializers, once unstable options such as a random uniform initialization have been removed, seem to provide similar qualities with a slight preference for the “glorot_normal” initialization procedure described in [46]. Concerning the parameters related to stopping criteria, when the number of epochs is very small the fit can be unstable, however after a certain threshold no big differences are observed. The stopping patience shows a very similar pattern: stopping too early can be disadvantageous but stopping too late does not seem to make a big difference. The positivity multiplier, however, shows a clear preference for bigger values. Finally, concerning the neural network architecture, a small number of layers seems to produce slightly better absolute results, however, one single hidden layer seems to lead to poor results. Concerning the activation functions, the hyperbolic tangent seems to be slightly preferred over the sigmoid. Once an acceptable hyperparameter setup has been achieved, a final fine tuning was performed, as some of the choices could have been biased by a bad combination of the other parameters.

Clearly, the result of the hyperoptimization depends on the underlying dataset: for instance, we have verified that hyperoptimization on a very large global dataset prefers a larger architecture. Therefore,

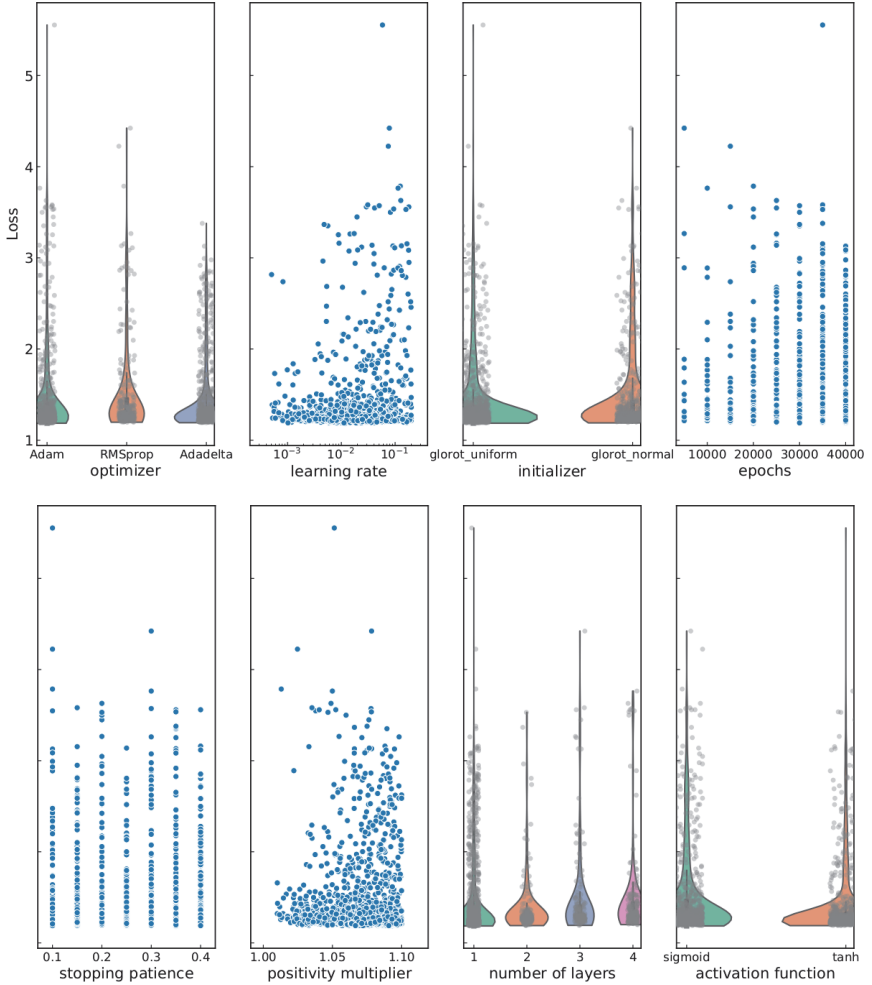


Fig. 10. Graphical representation of a hyperparameter scan for a DIS only fit with 2000 trials (from [42]). The loss function presented in the y -axis is an average of the validation and testing χ^2 . The shape of the violin plots represent a visual aid on the behavior of the fit as a function of the free parameter. Fatter plots represent better stability, i.e. configurations which are less likely to produce outliers.

the reliability and stability of the hyperoptimized methodology have to be checked a posteriori, as we will discuss in Sec. 3.4.

In summary, hyperoptimization has been implemented as a semi-automatic methodology, that is capable of finding the best

hyperparameter combination as the setup changes, e.g. with new experimental data, new algorithms or technologies.

3.3. Quality control

The hyperoptimization presented in Sec. 3.2 can be viewed as a meta-optimization in which the object of optimization is the methodology. This immediately raises the issue of quality control. In the fitting procedure, this is taken care by cross-validation, in which quality control is provided by the validation set. A similar quality control is now needed at the hyperoptimization level.

Indeed, if hyperoptimization is run by just optimizing on the validation figure of merit, a typical result is shown in Fig. 11, in which replicas for the up quark PDF for a hyperoptimized DIS fit are shown. It is clear that an unstable behavior is seen, characteristic of over-training. This can also be verified quantitatively: for example the value of the training χ^2 is much lower than that of the validation χ^2 . This may appear to be surprising, given that the hyperoptimization is performed on the validation χ^2 , while the training χ^2 is minimized in the fitting procedure. However, there inevitably exist correlations between the training and validation sets, for example

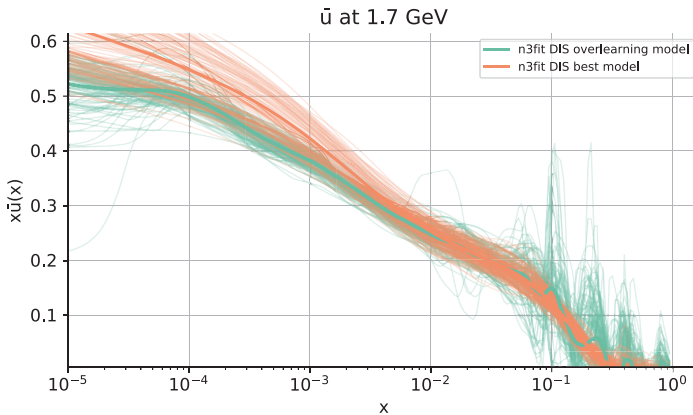


Fig. 11. Comparison of replicas for the up quark PDF obtained by hyperoptimized `n3fit` methodology without (green) and with (orange) quality control (from [42]).

through correlated theoretical and experimental uncertainties. Due to these correlations, hyperoptimization without quality control leads to overlearning.

The problem can be solved by introducing a testing set, which tests the generalization power of the model. The testing set is made out of datasets which are uncorrelated to the training and validation data, and none of which is used in the fitting either for training or validation. The test set plays the role of quality control for the hyperoptimization, as schematically summarized in Fig. 12.

Defining the best appropriate test dataset for PDF fits is particularly challenging due to the nature of the model regression through convolutions. Indeed, the choice of prescription for the test set presents a certain level of arbitrariness. For a first exploration, the test set has been constructed by utilizing datasets for which several experiments exist for the same process, and picking the experiment with smallest kinematic range. The corresponding data have been removed from training and validation, and used as a test set. A more refined option, which validates this first choice, will be discussed in Sec. 3.4.1 below.

We have applied this procedure both to DIS and global fits. The best models found in each case are compared in Table 3. For the global setup deeper networks are allowed without leading to overfitting. The hyperbolic tangent and the sigmoid functions are found to perform similarly. The initializer of the weights of the network,

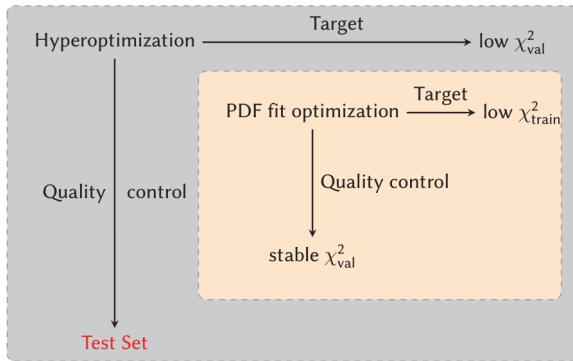


Fig. 12. Schematic overview of the hyperparameter quality control methodology.

Table 3. Best models found by our hyperparameter scan for the DIS and global setups using the new **n3fit** methodology.

Parameter	DIS only	Global
Hidden layers	2	3
Architecture	35-25-8	50-35-25-8
Activation	tanh	sigmoid
Initializer	glorot_normal	glorot_normal
Dropout	0.0	0.006
Optimizer	Adadelata	Adadelata
Max epochs	40000	50000
Stopping patience	30%	30%

Table 4. Comparison of the total χ^2 of the fit for both a DIS only and global fits found using the previous NNPDF3.1 and the new **n3fit** methodology.

	DIS only	Global
n3fit (new)	1.10	1.15
NNPDF3.1 (old)	1.13	1.16

however, carries some importance for the stability of the fits, with preference for the Glorot normal initialization method [46, 47] as implemented in Keras. Furthermore, adding a small dropout rate [48] to the hidden layers in the global fit reduces the chance of overlearning introduced by the deeper network, thus achieving more stable results. As expected, the bigger network shows a certain preference for greater waiting times (which also increases the stopping patience as is set to be a % of the maximum number of epochs). In actual fact, the maximum number of epochs is rarely reached and very few replicas are wasted.

Turning now to fit results, despite the significant difference in size and complexity of the dataset, the DIS and global fits perform similarly in describing the experimental data, as demonstrated by the χ^2 values presented in Table 4. It is interesting to compare results to

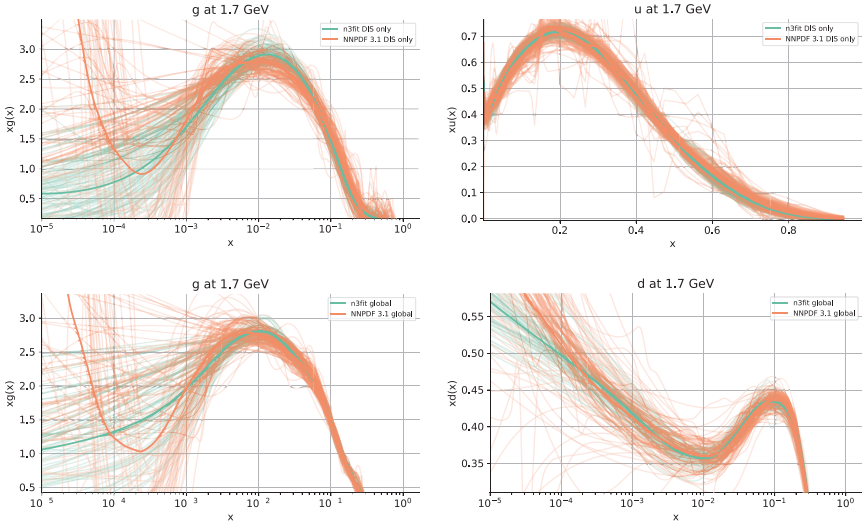


Fig. 13. Comparison of PDFs found using the previous NNPDF3.1 and the new **n3fit** methodology: for a DIS fit (top) the gluon (left) and up quark (right) are shown; for a global fit (bottom) the gluon (left) and down quark (right) are shown. (from [42]).

those obtained using the previous NNPDF3.1 methodology. The total χ^2 values are compared in Table 4: even though the new methodology leads to a slightly better fit, differences are small. PDF replicas obtained with either methodology (for the gluon and the up quark) are compared Fig. 13, both for the DIS and global fits. It is clear that the best-fit PDF, i.e. the average over replicas, is not much affected by the change in methodology (though somewhat smoother for **nnfit**).

A significant difference however is seen at the level of individual replicas: replicas found with the new methodology are rather more stable, i.e. they fluctuate rather less. This leads to slightly smaller uncertainties, and, more significantly, with the new methodology a smaller number of replicas is necessary in order to arrive to a stable average. The greater stability of the new methodology also leads to somewhat smaller uncertainties in the far extrapolation, i.e. in regions where there is no information and thus uncertainties are large: this is seen in Fig. 13 for the gluon distribution for $x \lesssim 10^{-4}$.

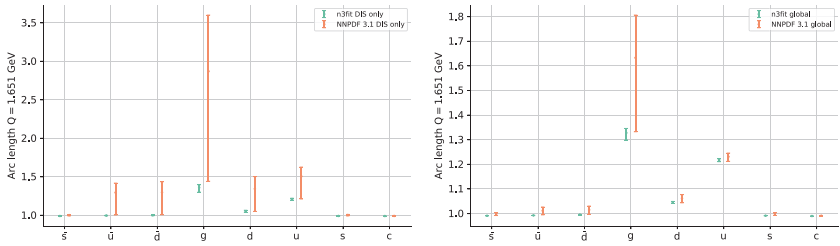


Fig. 14. Comparison of PDF arc-lengths found using the previous NNPDF3.1 and the new **n3fit** methodology in the DIS (left) and global (right) case. The mean and one- σ interval computed from a set of PDF replicas for each PDF is shown.

This raises the question of how to reliably assess uncertainties in extrapolation: we will return to this in Sec. 3.4.3 below.

A particularly transparent way of seeing this greater stability is to compare PDF arc-lengths. Because a PDF is a function of $0 < x < 1$, one may define the length of the curve traced by the PDF as x varies in this interval. A smoother PDF then has smaller arc-length. In Fig. 14, the mean and one- σ values of arclengths computed from a set of replicas with the new and old methodology are compared, both for the DIS and global fits. It is clear that, with the new methodology, the arc-length mean values are smaller, but especially the fluctuation of arc-length values between replicas is much smaller.

In summary, we conclude that the new hyperoptimized **n3fit** methodology leads to results which are in broad agreement with the current NNPDF3.1 methodology, thereby confirming that the latter is faithful and unbiased, as expected based on the closure tests of Sec. 2.3. However, thanks to code redesign and deterministic minimization it is possible to achieve greater computational efficiency, and thanks to the hyperoptimization it is possible to obtain, based on the same underlying datasets, more stable results (i.e. a smaller number of replicas is sufficient to achieve good accuracy) and somewhat smaller uncertainties. In short, the new **n3fit** methodology, while providing a validation of the current NNPDF methodology, displays greater computational efficiency, greater stability and greater precision without loss of accuracy. This in turn calls for more detailed validation and testing, as we now discuss.

3.4. *Validation and testing*

The `n3fit` methodology motivates and enables more detailed studies of fit quality. It enables them because thanks to its much greater computational efficiency it is now possible to perform rather more detailed explorations than it was possible with the previous slower methodology. It motivates them, because the goal of the new methodology is to allow for greater precision without loss of accuracy, namely, to extract more efficiently the information contained in a given dataset. It is then mandatory to make sure that no new sources of arbitrariness are introduced by the new methodology. Also, the new methodology is claimed to be more precise without loss of accuracy, i.e. to produce results which are more stable and have smaller uncertainty than the previous methodology given the same input. It is then crucial to perform validation tests which are sufficiently detailed that the validity of this claim can be tested: in practice, this means tests that are sufficiently detailed that the two methodologies can be distinguished, and that impose more stringent requirements on the methodology itself.

We will first discuss the new issue of robustness of the test-set methodology introduced in Sec. 3.3, then turn to a more detailed set of closure tests, similar to those of Sec. 2.3 but now exploiting the new methodology, and finally discuss a new kind of test of the generalization power of the methodology: “future testing”.

3.4.1. *Test-set stability*

One new source of ambiguity in the `n3fit` methodology is the choice of an appropriate test set. Indeed, the setup discussed in Sec. 3.3 was based on a particular choice of test set, but one would like to avoid as much as possible this kind of potentially biased subjective choice. Also, in that setup one has to discard some data from the dataset used for fitting and only include them in the test set. This contrasts with the desire to keep data in the training set as much as possible, in order to exploit as much as possible the (necessarily limited) dataset in order to determine the wide variety of features of the underlying PDFs.

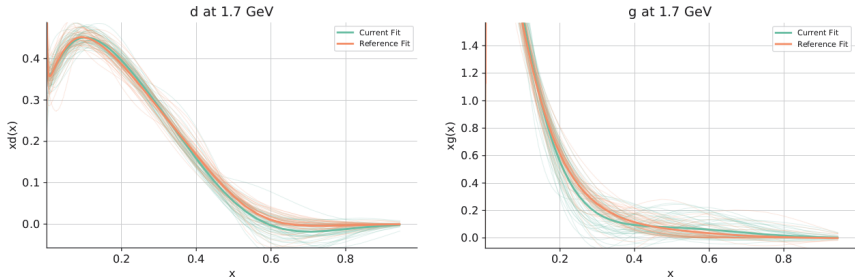


Fig. 15. Comparison between the best models from k -fold cross-validation (green) and manual selection (red)[49].

These goals can be achieved through a k -fold cross-validation. In this algorithm, data are subdivided into k partitions, each of which reproduces the broad features of the full dataset. Each of the partitions then plays in turn the role of the test set, by being excluded from the fit. A variety of figures of merit can then be chosen for hyperparameter optimization, such as the mean value of the loss over excluded partitions, or the best worst value of the validation loss of the excluded partition.

This k -folding procedure has been implemented, and stability upon different choices of hyperoptimization figure of merit has been explicitly checked. Results are shown in Fig. 15, where the best PDF models estimated using k -folding are compared to those obtained through the simple test-set procedure of Sec. 3.3. Similar results are found using either method. While confirming the reliability of the manually selected method of Sec. 3.3, this allows us to replace it with the more robust and unbiased k -folding method.

3.4.2. Closure testing

We now turn to closure testing, as presented in Sec. 2.3 in the context of NNPDF3.0 [24]. We have applied the closure testing methodology of Sec. 2.3, but now using the `n3fit` methodology and the more recent and wider NNPDF3.1 [12] dataset and theory settings. Hence, level 2 data are now in one-to-one correspondence with data in the NNPDF3.1 dataset, and, more importantly, we can take advantage of the greater computational efficiency of `n3fit`.

A first example of this is that it is now possible to perform confidence level tests based on actual full reruns. Indeed, recall from Sec. 2.3 that a computation of a closure test confidence level requires producing several independent fits, each with a sufficiently large number of replicas, so that the population of central values and uncertainties in each fit can be compared to an underlying truth. Thanks to the use of `n3fit`, it has now been possible to perform 30 different closure test level 2 fits, each with 40 replicas [50]. Results are then further enhanced and stabilized by using bootstrapping, i.e. by drawing random subsets of fits and random subsets of replicas from each fit and computing the various estimators for the resample of fits and replicas. It has been possible to check in this way that results are essentially stable with at least 10 fits with at least 25 replicas each, in that increasing the number of fits and replicas results are unchanged. All numbers quoted below refer to results obtained with the largest numbers of fits and replicas. The fact that such a relatively small number of replicas is sufficient to achieve stable result is a reflection of the greater stability of `n3fit` replicas discussed in Sec. 3.4.

As a first test, we recompute the histogram of deviations of Fig. 7, but now using NNPDF3.1 data. We can now compare the histogram actually computed using 30 fits with 40 replicas each, with the histogram approximately determined using a single 100 replica level 2 fit and 100 single-replica level 1 fits, as it was done for Fig. 7 (labeled “NNPDF3.1 methodology”). The result is shown in Fig. 16. It is clear that the validation is successful also for the (rather wider) NNPDF3.1 dataset: the one- σ confidence level is now equal to 65%, and the mean of the histogram is now essentially unbiased, unlike in Fig. 7 where a small bias was present. Also the approximate method used in Sec. 2.3 and [24] is reasonably accurate: specifically, the true value 65% is reasonably well approximated by the value 71% found using the approximate method.

We can now proceed to more detailed closure tests by computing confidence levels more extensively. A useful tool in this context is the bias-variance ratio. This, for Gaussian distributions, contains exactly the same information as the one- σ confidence level of predicted values

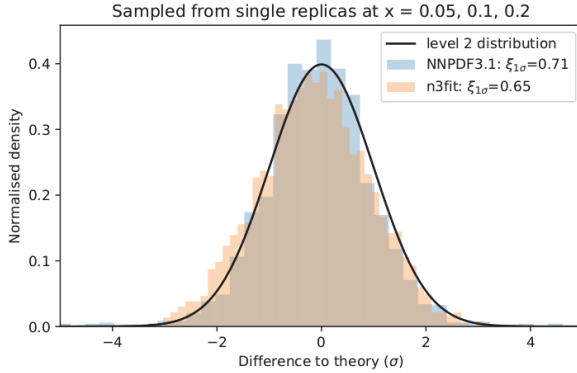


Fig. 16. Same as Fig. 7, but now using NNPDF3.1 data and methodology, and comparing results obtained using the approximate methodology of Sec. 2.3 (NNPDF3.1 methodology) and the exact methodology (`n3fit` methodology)[50].

with respect to the underlying truth considered in Sec. 2.3. For uncorrelated data, the bias-variance ratio is defined as the mean square deviation of the prediction from the truth (bias), divided by the expected one- σ uncertainty (variance). The square-root of the bias-variance ratio

$$R_{bv} = \sqrt{\frac{1}{N_{\text{dat}}} \sum_{i=1}^{N_{\text{dat}}} \frac{(d_i - d_i^{(0)})^2}{\sigma_i^2}} \quad (11)$$

(where d_i , σ_i and $d_i^{(0)}$ are respectively the prediction, uncertainty and true value for the i th datapoint) is the ratio between observed and predicted uncertainties, and thus it should be equal to one for a perfect fit. The generalization to the correlated case is straightforwardly obtained by expressing the numerator and denominator under the square root in Eq. (11) in terms of the covariance matrix. We have verified explicitly that the value of the one- σ confidence level interval computed using the measured bias-variance ratio coincides with the measured confidence level, within statistical accuracy, so either can be equivalently used.

We can now turn to more detailed comparisons. First, the comparison can be done for each PDF individually, rather than for all PDFs lumped together. Second, the comparison can also be done at

the level of experimental data: namely, instead of determining the deviation between the fitted and true PDF we determine the deviation between the prediction obtained using the best-fit PDF and the true PDF for each of the datapoints in the NNPDF3.1 dataset.

It should be noted that, of course, the predictions for individual datapoints are correlated due to the use of common underlying PDFs, with correlations becoming very high for datapoints which are kinematically close, so that the integral Eq. (1) is almost the same. These correlations can be simply determined by computing the covariance matrix between all datapoints induced by the use of the underlying PDFs, which in turn is done by determining covariances over the PDF replica sample. Confidence levels are then determined along eigenvectors of this covariance matrix, and can be compared to the bias-variance ratio, either by using its general form in the non-diagonal data basis, or equivalently, using Eq. (11), but with the sum running not on the original datapoints, but rather over the eigenvectors of the covariance matrix.

Of course, the PDFs themselves are also correlated. The histograms in Figs. 7 and 16 were computed by sampling each PDF at three widely spaced points in x so as to minimize this correlation, but of course computing a histogram of deviations with correlations neglected is still an approximation. When performing comparisons in PDF space we have now therefore also computed the covariance between PDFs over the replica sample, and determined confidence intervals along its eigenvectors, and the corresponding bias-variance ratio values with correlations kept into account.

A first comparison has been performed by computing the bias-variance ratio at the data level. This leads to an interesting result. Recall from Sec. 2.3 and Fig. 5 that the total PDF uncertainty consists of three components of comparable size, the first of which is due to the need to interpolate between data. Clearly, this latter component is absent if one compares the prediction to the same data which have been used to produce the PDF set. Indeed, we find that the square root of the bias-variance ratio computed for the NNPDF3.1 dataset (more than 4000 datapoints) is $R_{bv} = 0.74$. If we compute the same ratio for a new wide dataset including about 1300 HERA,

LHCB, ATLAS and CMS data not used in the fit we find that the value is $R_{bv} = 0.9$. The difference between these two values can be understood as an indication of the fact that in the former case the bias does not include the level 1 uncertainty, while the variance (which should be used for new prediction) does. The value $R_{bv} = 0.9$ means that PDF uncertainties on predictions are accurate to 10% (and somewhat overestimated).

We next computed both the bias-variance ratio and the one-sigma confidence level at the PDF level. PDFs have been sampled at four points for each PDF, in a region in x corresponding to the data region, and the covariance matrix has been subsequently diagonalized as discussed above. Results are shown in Table 5 for individual PDF combinations. It is clear that, especially for the PDF combinations that are known with greater accuracy, such as the quark singlet Σ and the gluon g , uncertainties are faithful: only the combination $T8$ which measures the total strangeness shows a certain amount of uncertainty underestimation, by about 30%.

Table 5. The bias-variance ratio R_{bv} (Eq. (11)) and the one-sigma confidence level for individual PDFs, computed using four points in x space per PDF along eigenvectors of the covariance matrix [50].

PDF	R_{bv}	one- σ c.l.
Σ	0.9	70%
gluon	0.9	69%
V	1.0	66%
V3	1.0	93%
V8	0.9	71%
T3	0.6	89%
T8	1.3	46%
total	0.9	0.71

3.4.3. Chronological future tests

The closure tests essentially verify the reliability of results in the data region. A much more difficult task is to verify the power of generalization of the methodology: namely, whether PDFs determined with a subset of data are able to correctly predict the behavior of new data, including those that extend the kinematic domain used for PDF determination. In practice, this means testing whether PDF uncertainties are reliable also in regions in which they start growing significantly because of lack of information.

This is done by “chronological” or “future” tests. Namely, we consider an existing (or hypothetical) past dataset, we train PDFs based on it, and we compare the best-fit results with later data which extend the kinematic region. A first test of this kind has been performed only including data which predated the HERA electron–proton collider, and which thus approximately correspond to the information on PDFs available around 1995. This is especially interesting since it is well known (see e.g. [51]) that the best-fit gluon shape substantially changed after the advent of HERA data, as pre-HERA data impose only very loose constraints on the gluon PDF.

We have thus produced a PDF determination using `n3fit` methodology, but only including pre-HERA data, and now performing a dedicated hyperparameter optimization based on this restricted dataset. The best-fit gluon determined in this way is compared to the current best-fit gluon in Fig. 17. Some subsequent data which are sensitive to the gluon, specifically the proton structure function F_2 , which is sensitive to the gluon at small x , and top-pair production at the LHC, which is sensitive to the gluon at medium-high x , are compared to predictions obtained using this PDF set in Fig. 18.

It is clear that the test is successful. In the region $x \lesssim 0.15$, where the gluon is currently known accurately thanks to HERA data, but it is extrapolated when only using pre-HERA data, the uncertainty grows very large, yet the two fits are compatible within these large uncertainties, and the new data are within the uncertainty of the extrapolated prediction. This is a highly non-trivial test of the generalizing power of the hyperoptimized `n3fit` methodology. Note also

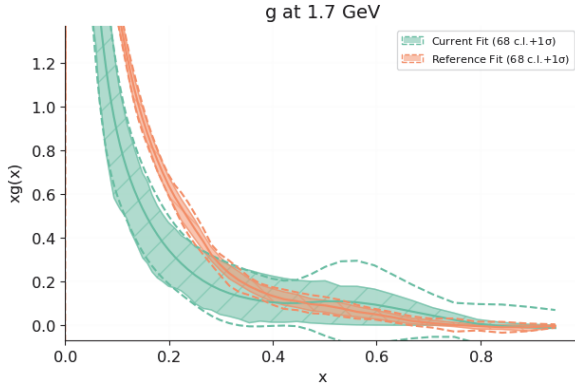


Fig. 17. The gluon PDF determined using pre-HERA data (green) compared to the current best-fit (orange) [49].

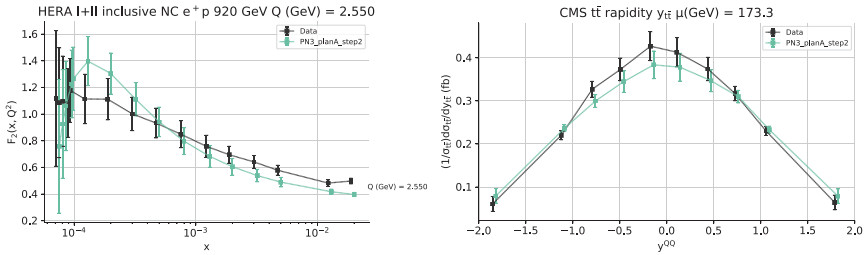


Fig. 18. Data for the proton structure function f_2 measured at HERA (left) and top-pair production measured at the LHC (right) compared to a prediction based on PDFs determined from a fit to pre-HERA data [49].

that this provides us with a test of the stability of the hyperoptimized methodology, in that it means that a methodology hyperoptimized to the much larger current dataset leads to reliable results even when used on the much more restrictive past dataset.

The optimization of the generalization power of our methodology is at the frontier of our current understanding and remains a challenging open problem.

3.5. Outlook

The `n3fit` methodology will be used in the construction of future PDF releases, starting with the forthcoming NNPDF4.0 PDF set.

The greater efficiency of this methodology will be instrumental in dealing with an ever increasing dataset, while its greater accuracy will be instrumental in reaching the percent-level uncertainty goal which is likely required for discovery at the HL-LHC [25]. Avenues of research for future methodological developments which are currently under consideration include the possibility of an integrated reinforcement learning framework for the development of an optimal PDF methodology, the exploration of machine learning tools alternative to neural networks, such as Gaussian processes, the exploration of inference tools, such as transfer learning, for the modeling of theoretical uncertainties, and a deeper understanding of the generalizing power of the methodology outside the data region.

Acknowledgments

We thank all the past and current members of the NNPDF collaboration and the members of the N3PDF team, whose work is behind most of the results reported here, for collaboration and for countless stimulating discussions, and in particular Rosalyn Pearson for a reading of the first draft. We acknowledge financial support from the European Research Council under the European Union's Horizon 2020 research and innovation Programme (grant agreement no. 740006).

References

- [1] R. McElhaney and S. F. Tuan, Some consequences of a modified Kuti Weiskopf quark parton model, *Phys. Rev. D* **8** (1973) 2267.
- [2] T. Kawaguchi and H. Nakkagawa, Analysis of scaling violation in terms of theories with anomalous dimensions, KUNS 380 (1976).
- [3] A. De Rujula, H. Georgi and H. D. Politzer, Demythification of electroproduction, local duality and precocious scaling, *Ann. Phys.* **103** (1977) 315.
- [4] P. W. Johnson and W.-K. Tung, Comparison of asymptotically free theories with high- energy deep inelastic scattering data, *Nucl. Phys. B* **121** (1977) 270.
- [5] M. Gluck and E. Reya, Operator mixing and scaling deviations in asymptotically free field theories, *Phys. Rev. D* **14** (1976) 3034.
- [6] I. Hinchliffe and C. H. Llewellyn Smith, Detailed treatment of scaling violations in asymptotically free gauge theories, *Nucl. Phys. B* **128** (1977) 93.

- [7] S. Forte, L. Garrido, J. I. Latorre and A. Piccione, Neural network parametrization of deep inelastic structure functions, *J. High Energy Phys.* **05** (2002) 062; arXiv:hep-ph/0204232.
- [8] NNPDF, L. Del Debbio, S. Forte, J. I. Latorre, A. Piccione and J. Rojo, Unbiased determination of the proton structure function F_2^p with faithful uncertainty estimation, *J. High Energy Phys.* **03** (2005) 080; arXiv:hep-ph/0501067.
- [9] NNPDF, L. Del Debbio, S. Forte, J. I. Latorre, A. Piccione and J. Rojo, Neural network determination of parton distributions: The nonsinglet case, *J. High Energy Phys.* **03** (2007) 039; arXiv:hep-ph/0701127.
- [10] NNPDF, R. D. Ball, L. Del Debbio, S. Forte, A. Guffanti, J. I. Latorre, A. Piccione, J. Rojo and M. Ubiali, A Determination of parton distributions with faithful uncertainty estimation, *Nucl. Phys. B* **809** (2009) 1, arXiv:0808.1231 [hep-ph]. [Erratum: *Nucl. Phys. B* **816**, (2009) 293].
- [11] R. D. Ball, L. Del Debbio, S. Forte, A. Guffanti, J. I. Latorre, J. Rojo and M. Ubiali, A first unbiased global NLO determination of parton distributions and their uncertainties, *Nucl. Phys. B* **838** (2010) 136; arXiv:1002.4407 [hep-ph].
- [12] NNPDF, R. D. Ball *et al.*, Parton distributions from high-precision collider data, *Eur. Phys. J. C* **77** (2017) 663; arXiv:1706.00428 [hep-ph].
- [13] R. K. Ellis, W. J. Stirling and B. R. Webber, *QCD and Collider Physics* (Cambridge University Press, 1996).
- [14] S. Forte, Parton distributions at the dawn of the LHC, *Acta Phys. Polon. B* **41** (2010) 2859; arXiv:1011.5247 [hep-ph].
- [15] J. Gao, L. Harland-Lang and J. Rojo, The structure of the proton in the LHC precision era, *Phys. Rept.* **742** (2018) 1; arXiv:1709.04922 [hep-ph].
- [16] J. J. Ethier and E. R. Nocera, Parton distributions in nucleons and nuclei, *Ann. Rev. Nucl. Part. Sci.* (2020) 1; arXiv:2001.07722 [hep-ph].
- [17] H.-W. Lin *et al.*, Parton distributions and lattice QCD calculations: toward 3D structure, preprint (2020); arXiv:2006.08636 [hep-ph].
- [18] R. G. Roberts, *The Structure of the Proton: Deep Inelastic Scattering* (Cambridge University Press, 1990).
- [19] CTEQ, H. L. Lai *et al.*, Global QCD analysis of parton structure of the nucleon: CTEQ5 parton distributions, *Eur. Phys. J. C* **12** (2000) 375; arXiv:hep-ph/9903282.
- [20] T.-J. Hou *et al.*, New CTEQ global analysis of quantum chromodynamics with high-precision data from the LHC (2019); arXiv:1912.10053 [hep-ph].
- [21] D. Stump, J. Pumplin, R. Brock, D. Casey, J. Huston, J. Kalk, H. Lai and W. Tung, Uncertainties of predictions from parton distribution functions. 1. The Lagrange multiplier method, *Phys. Rev. D* **65** (2001) 014012; arXiv:hep-ph/0101051.
- [22] J. Pumplin, D. Stump, R. Brock, D. Casey, J. Huston, J. Kalk, H. Lai and W. Tung, Uncertainties of predictions from parton distribution functions. 2. The Hessian method, *Phys. Rev. D* **65** (2001) 014013; arXiv:hep-ph/0101032.

- [23] A. D. Martin, R. G. Roberts, W. J. Stirling and R. S. Thorne, Uncertainties of predictions from parton distributions. I: Experimental errors, *Eur. Phys. J. C* **28** (2003) 455; arXiv:hep-ph/0211080.
- [24] NNPDF, R. D. Ball *et al.*, Parton distributions for the LHC Run II, *J. High Energy Phys.* **04** (2015) 040; arXiv:1410.8849 [hep-ph].
- [25] P. Azzi *et al.*, Report from working group 1: Standard model physics at the HL-LHC and HE-LHC (2019); arXiv:1902.04070 [hep-ph].
- [26] E. Bagnaschi and A. Vicini, A new look at the estimation of the PDF uncertainties in the determination of electroweak parameters at hadron colliders (2019); arXiv:1910.04726 [hep-ph].
- [27] NNPDF, J. Rojo, R. D. Ball, L. Del Debbio, S. Forte, A. Guffanti, J. I. Latorre, A. Piccione and M. Ubiali, Update on neural network parton distributions: NNPDF1.1, in *38th Int. Symp. Multiparticle Dynamics* (2009); arXiv:0811.2288 [hep-ph].
- [28] NNPDF, R. D. Ball, L. Del Debbio, S. Forte, A. Guffanti, J. I. Latorre, A. Piccione, J. Rojo and M. Ubiali, Precision determination of electroweak parameters and the strange content of the proton from neutrino deep-inelastic scattering, *Nucl. Phys. B* **823** (2009) 195; arXiv:0906.1958 [hep-ph].
- [29] V. Bertone, S. Carrazza and N. P. Hartland, APFELgrid: A high performance tool for parton density determinations, *Comput. Phys. Commun.* **212** (2017) 205; arXiv:1605.02070 [hep-ph].
- [30] NNPDF, R. D. Ball, L. Del Debbio, S. Forte, A. Guffanti, J. I. Latorre, J. Rojo and M. Ubiali, Fitting parton distribution data with multiplicative normalization uncertainties, *J. High Energy Phys.* **05** (2010) 075; arXiv:0912.2276 [hep-ph].
- [31] R. Abdul Khalek *et al.*, A first determination of parton distributions with theoretical uncertainties (2019); arXiv:1905.04311 [hep-ph].
- [32] NNPDF, R. Abdul Khalek *et al.*, Parton distributions with theory uncertainties: general formalism and first phenomenological studies, *Eur. Phys. J. C* **79** (2019) 931; arXiv:1906.10698 [hep-ph].
- [33] M. D. Zeiler, ADADELTA: An adaptive learning rate method (2012); arXiv:1212.5701 [cs.LG].
- [34] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2014); arXiv:1412.6980 [cs.LG].
- [35] R. D. Ball *et al.*, Parton distributions with LHC data, *Nucl. Phys. B* **867** (2013) 244; arXiv:1207.1303 [hep-ph].
- [36] L. Demortier, Open issues in the Wake of Banff 2011 in *Proc. PHYSTAT 2011 Workshop on Statistical Issues Related to Discovery Claims in Search Experiments and Unfolding*, CERN, Geneva, Switzerland 17–20 January 2011 (2011).
- [37] A. D. Martin, W. J. Stirling, R. S. Thorne and G. Watt, Parton distributions for the LHC, *Eur. Phys. J. C* **63** (2009) 189; arXiv:0901.0002 [hep-ph].
- [38] G. Cybenko, *Math. Control Signal Syst.* **2** (1989) 303.
- [39] F. Chollet *et al.*, Keras (2015); <https://keras.io>.

- [40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems (2015); <http://tensorflow.org/>.
- [41] J. Bergstra, D. Yamins and D. D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in *Proc. 30th Int. Conf. International Conference on Machine Learning, ICML'13*, Vol. 28 (2013).
- [42] S. Carrazza and J. Cruz-Martinez, Towards a new generation of parton densities with deep learning models, *Eur. Phys. J. C* **79** (2019) 676; arXiv:1907.05075 [hep-ph].
- [43] T. Tieleman and G. Hinton, Lecture 6.5 — RmsProp: Divide the gradient by a running average of its recent magnitude, in *Neural Networks for Machine Learning* (Coursera, 2012).
- [44] V. Bertone, S. Carrazza and J. Rojo, APFEL: A PDF evolution library with QED corrections, *Comput. Phys. Commun.* **185** (2014) 1647; arXiv:1310.1394 [hep-ph].
- [45] J. Bergstra, R. Bardenet, Y. Bengio and B. Kégl, Algorithms for hyperparameter optimization, in *Proc. 24th Int. Conf. Neural Information Processing Systems, NIPS'11* (Curran Associates Inc., USA, 2011).
- [46] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feed-forward neural networks, in *Proc. Int. Conf. Artificial Intelligence and Statistics (AISTATS10)* (Society for Artificial Intelligence and Statistics, 2010).
- [47] Y. Bengio and X. Glorot, Understanding the difficulty of training deep feed forward neural networks, in *Int. Conf. Artificial Intelligence and Statistics* (2010).
- [48] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors (2012); arXiv:1207.0580.
- [49] NNPfDf, In preparation.
- [50] L. Del Debbio and M. Wilson, In preparation.
- [51] W.-K. Tung, Status of global QCD analysis and the parton structure of the nucleon, in *12th Int. Workshop on Deep Inelastic Scattering (DIS 2004)* (2004); arXiv:hep-ph/0409145.

Part VIII

**Scientific Competitions
and Open Datasets**

This page intentionally left blank

Chapter 20

Machine Learning Scientific Competitions and Datasets

David Rousseau^{*,†} and Andrey Ustyuzhanin^{†,§}

** Université Paris-Saclay, CNRS/IN2P3,
IJCLab, 91405 Orsay, France*

*† National Research University Higher School of Economics
and Yandex School of Data Analysis, Moscow, Russia*

‡ david.rousseau@ijclab.in2p3.fr

§ andrey.u@gmail.com

A number of scientific competitions have been organized in the last few years with the objective of discovering innovative techniques to perform typical high-energy physics tasks, like event reconstruction, classification and new physics discovery. Four of these competitions are summarized in this chapter, from which guidelines on organizing such events are derived. In addition, a choice of competition platforms and available datasets are described.

1. Introduction

Competitions play an important role in the development of Machine Learning algorithms. The 2012 breakthrough of a Convolutional Neural Network [1] in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition on labeling objects in the ImageNet dataset is often indicated to be the start of the “Deep Learning revolution”. The ImageNet dataset itself is considered as a standard candle in countless papers, as well as for teaching and training.

Competitions in high-energy physics are much less part of the culture. One explanation for this is that sharing data has not been the

norm, although this is changing. Collaborations of scientists usually analyze the data from the experiment they have built and share the result of the analyses but not the data itself.

For a specific task, e.g. particle identification or event classification, one can find papers on algorithm A applied on dataset alpha; algorithm B applied on dataset beta. The metric used will be similar, at best, identical. However, suppose one sees better result on one side. In that case, it is difficult to infer if algorithm A is intrinsically better than algorithm B or that dataset alpha makes the task easier. If one wants to find a better algorithm, one would gather papers, go to workshops, talk to experts to have suggestions of better algorithms that one would have to re-implement to apply on one's dataset. The difficulty is not just about acquiring the software but also the accompanying expertise.

Scientific competitions are an alternative approach structured around so-called Common Task Framework (CTF) [2] that involves:

- (1) A publicly available training dataset involving, for each observation, a list of feature measurements, and a class label for that observation;
- (2) A set of enrolled competitors whose common task is to infer a class prediction rule from the training data;
- (3) A scoring referee, to which competitors can submit their prediction rule. The referee runs the prediction rule against a testing dataset which is sequestered behind a screen. The referee objectively and automatically reports the score (e.g. prediction accuracy) achieved by the submitted rule.

In reality, the scoring might be quite complicated as translation between domain challenge requirements to a straightforward computational form requires both fluent speaking and understanding potential flaws of both: the domain and machine learning languages.

The objective of this chapter is to look under the hood of scientific competitions and encourage participation and foster the organization of future competitions.

The chapter is organized as follows. Section 2–5 give a summary of four physics competitions and related datasets: Sec. 2 HiggsML on

event classification, Sec. 3 Flavor of Physics on event classification in the presence of mismodelings, Sec. 4 TrackML on pattern recognition, Sec. 5 LHC-Olympics on anomaly detection. Section 6 lists available competition platforms and Sec. 7 lists available open datasets. Section 8 indicates general guidelines for scientific competition organizers, based on the experience organizing such competitions. Section 9 is the conclusion.

2. HiggsML

The Higgs Boson Machine Learning challenge (HiggsML in short)^a took place on the Kaggle platform in 2014. At the time, Machine Learning was already used at the LHC experiments (see [3] although some of the results quoted there are post-2014) but in most cases, this was Boosted Decision Trees, while the Deep Learning revolution had already started elsewhere. Some breakthrough papers (in particular [4], see also Chapter 3) were indicating a potential for Deep Learning for final analysis. The motivation for the HiggsML challenge was to reach out to the Computer Science community to explore the possibilities of modern Machine Learning algorithms for a classification problem pertaining to Higgs boson physics at the LHC. The HiggsML challenge is described in details in [5], which is the document accompanying the final release of the dataset [6] on the CERN Open Data Portal; lessons derived from the challenge are described in [7] with more details in the write-up of contributions of the dedicated HEPML workshop which has taken place at NeurIPS 2014 [8]. Only a summary is given here.

2.1. Dataset and score

A dataset of 250,000 events (out of an original dataset of about 800k events, the complement being held out for evaluation) was provided, each event providing 30 features, measurements from simulation proton collision, which had been used by the ATLAS experiment for its first paper on the specific topic [9]. The events were either from Higgs

^a<https://higgsml.lal.in2p3.fr>.

boson decay into a tau-lepton pair (the signal) or from other processes, Z -boson decaying into a tau-lepton pair as well, top pairs and W . There were two classes of features, the primary and the derived ones. The primary ones were essentially the 3-momentum of key event particles: an electron or muon, a τ hadron decay, the missing transverse energy which is a 2D pseudo-particle, and the possible leading and subleading jets. The derived parameters are features (which could be recomputed from the primary ones) defined in the same ATLAS paper [9], that offer a good separation between signal and background. These derived features had been crafted by physicists to maximize the separation between signal and background. It should be noted that not all events have two jets (some have zero, some have one) so that the jet quantities (primary or derived) might be absent for some events. For training, in addition to the label “S” for signal or “B” for background, a weight is also given, which allows computing the expected number of signal and background events (for 2012 LHC luminosity).

A non-standard (for Machine Learning) figure of merit was used to rank the criterion, the Approximate Median Significance (AMS), which quantify (in number of standard deviations) the expected discovery significance of an experiment. It is obtained with the following formula where s (respectively, b) is the number of expected signal (respectively background) events:

$$AMS = \sqrt{2} \sqrt{(s + b + 10) \log(1 + s/(b + 10)) - s}, \quad (1)$$

where s (respectively, b) is the sum of the weights of signal (respectively, background) events passing the selection in the test sample. This is the usual (for physicists) so-called Asimov formula [10], except that 10 is added to b as a regularization term, to avoid nonphysically large significance in the very strong selection regime, where b can be less than 1. The impact of using this figure of merit compared to, e.g. a more classical ROC-AUC or accuracy criteria is that more importance is given to the part of the ROC curve with large background rejection and small signal efficiency (large True Negative and small True Positive), see Sec. 3.3.3 for a different means to achieve the same goal.

2.2. Competition

The participation was large with close to 2000 participants, a record at the time for Kaggle challenges. The ranking among the top 10 was tight, with some rank changes when the private leaderboard (established on a preserved dataset) was revealed. Subsequent studies with a bootstrap technique showed that number 1 (Gabor Melis) rank was solid, while number 2 (Tim Salimans) and number 3 (nhlx5haze) could have exchanged places, but were well separated from number 4 and beyond (see Fig. 1). Figure 2 shows the AMS performance for some top participants. Particularly, interesting curves are the ones from Lubos Motl's Team who was number 1 on the public leaderboard but fell to number 8 on the final leaderboard. A sharp peak on the public test curve (with no counterpart on the private test curve) is due to public leaderboard overfitting as the team has claimed to “play” the public leaderboard, adjusting parameters in a semi-automatic fashion to improve their public score.

One key feature provided was DER_{mass}_MMC, which is an estimator of the mass of the $\tau^+\tau^-$ pair obtained through a complex MCMC estimation. Although all the inputs to do this calculation

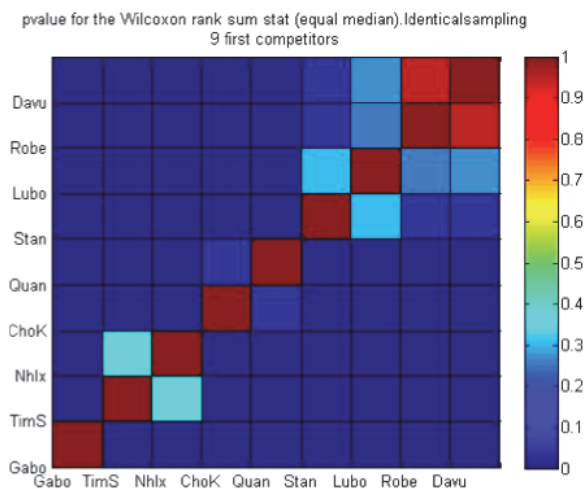


Fig. 1. p -Values of the pairwise Wilcoxon rank sum test (from [7]).

were available, the software to compute it has not been released. Nevertheless, **kesterlester**, a physicist, has released the result of a similar computation under name **Cake**.^b It improved significantly the results of most participants, but the top ones did not see any improvements using it, most likely because their classifiers were already able to extract sufficient information from the features they had built.

The number 1 **Gabor Melis** used an ensemble of dense neural network with three hidden layers; however, from his assessment, he got an edge through careful use of nested Cross-Validation. Figure 2(a) shows his AMS performance on the public test set to be relatively flat compared to others, while Fig. 2(d) shows it is clearly above

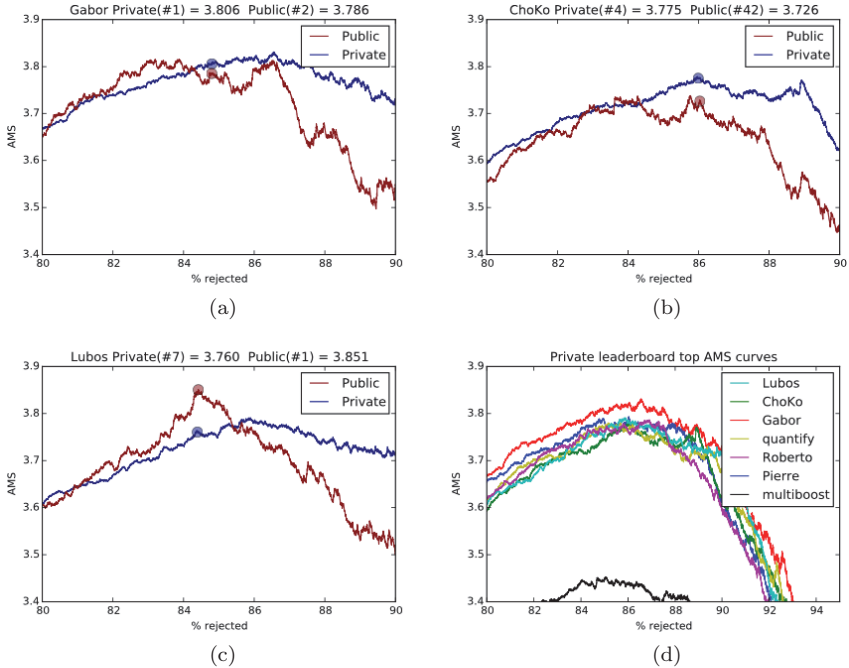


Fig. 2. AMS curves for some participants comparing performance between the public and private test sample (a)–(c) and for several participants on the private test sample (d). The horizontal axis is the weighted proportion of selected background events (from [7]).

^b<https://www.kaggle.com/c/higgs-boson/discussion/10329>.

the others almost everywhere. Number 2 **Tim Salimans** has used an ensemble of Regularized Greedy Forest and number 3 **nh1x5haze** an ensemble of neural networks. Tianqi Chen and Bing Xu (team **crowwork**) have reached a modest rank 45, however, they got the special Hep ML prize from the jury, for they have released early in the competition their new Boosted Decision Tree algorithm XGBoost [11] and supported its use. XGBoost was used by many participants including top 10. XGBoost popularity has grown ever since and has been one of the primary tools used in Kaggle competitions [12]. Its usage is also growing in high-energy physics publications (e.g. [13]), given its high speed and classification performance.

2.3. *Follow-up studies*

The complete HiggsML dataset has been released on the CERN Open Data portal [6]. The full 818,238 events dataset has been released. The team has hesitated to release it completely, without holding a reserve test set. The benefit is that future users have the largest statistics (compared to the 250,000 Kaggle participants had). The downside is that there is no possibility for independent check for overfitting.

The HiggsML dataset (either from Kaggle or from CERN ODP) has been used extensively since the competition for various tutorials (e.g. [14]), courses (e.g. [15]), blog posts (e.g. [16]), PhD dissertations (e.g. [17]) and papers (see later in this section). While most are informative, some common mistakes have been seen, especially concerning the weight:

- as the weights are such that events are normalized to 2012 Large Hadron Collider data taking, an event weight reflects the way it was generated so that the weight is an almost certain give away of the signal or background label of the event. For this reason, during the Kaggle competition, it was only provided for the training sample, not the test sample. However, in some follow-up studies, some people have used the weight as a regular feature (although being strongly advised against it through the accompanying document) which give them extremely good performances. Since many people

do not read the documentation, probably the weight should have been renamed `weight_DO_NOT_USE_IN_TRAINING`

- in contrast with AUC or accuracy, the AMS does depend on the total weight of each sampling (in a trivial way $\text{AMS} \simeq \frac{s}{\sqrt{b}}$, so simply using half of the dataset divides AMS by $\sqrt{2}$). The dataset documentation does specify that whenever a subsample of the dataset is used for evaluation, weights should be scaled up by the inverse of the fractional size of the subsample (so a factor 2 for a subsample of $\frac{1}{2}$). This was not done correctly in some follow-up studies.

Besides, claims for AMS well above 3.81 reached by the winner of the competition are most likely due to overtraining.

A recent thorough post-challenge analysis was done [18], where the author has studied data augmentation, learning rate and momentum scheduling, (advanced) ensembling in both model-space and weight-space, and alternative architectures and connection methods, using a modern NN library. He reaches the same 3.81 AMS although with considerably faster training time.

Beyond classification, a python script allows introducing systematic effects (miscalibration or poorly known backgrounds) [19] and has been used to investigate how to deal with systematic effects [20–23], see also Chapter 17 in this book.

3. Flavor of Physics

3.1. Introduction

Offline data analysis in particle physics has many challenges that can provoke communications between the physics and data science communities. In addition to sensitivity increase, there are questions of (a) training ML algorithms using the mixture of real and simulated samples (see Sec. 3.3.1) and (b) reducing the impact of so-called *nuisance parameters* that affect the likelihood and posterior distributions non-systematically (see Sec 3.3.2). LHCb collaboration^c

^cMain contributors: Thomas Blake, Marc-Olivier Bettler, Marcin Chrzaszcz, Francesco Dettori, Andrey Ustyuzhanin and Tatiana Likhomanenko.

prepared a competition to address those challenges via a competition on Kaggle platform^d that was active for three months in 2015.

3.1.1. *The challenge goal*

The main goal of this challenge is to gain sensitivity in the search for $\tau^- \rightarrow \mu^- \mu^- \mu^+$ decays. That is achieved by improving the discriminating power between signal events (where the decay did occur) and background events (where it did not). LHCb collaboration provides signal and background samples for training and testing. The evaluation happens in three steps: firstly, the classifier is checked not to depend too strongly on the discrepancies between real data and simulation. Then it checks if the classifier output is decorrelated with the τ mass. Finally, the comparison of the classifiers is performed using the weighted area under the ROC curve.

3.1.2. *Physics motivation*

The search for decays that do not conserve primary particle flavor quantities started in the late 1930s with the discovery of the muon (μ). It was believed that muons were an excited electron state in which case one would expect to observe a decay $\mu^- \rightarrow e^- \gamma$, where a photon with predictable energy would be emitted. No such process has ever been observed. The muon decays instead through the process $\mu^- \rightarrow e^- \nu_\mu \bar{\nu}_e$, with the emission of a muon neutrino and an electron anti-neutrino to preserve the total electronic and muonic lepton numbers. Similarly, in the 1970s an even heavier lepton was discovered as product of $e^+ e^-$ annihilation: the tau (τ) lepton, with a mass equivalent to about 3500 electrons. Typical decays of the τ leptons are $\tau^- \rightarrow e^- \nu_\tau \bar{\nu}_e$ and $\tau^- \rightarrow \mu^- \nu_\tau \bar{\nu}_\mu$, that conserve the various lepton numbers involved. However, if lepton flavor is not a perfectly conserved quantity in nature, and various explanations of the matter asymmetry in the universe require this, then the τ lepton can also decay into three muons though the reaction $\tau^- \rightarrow \mu^- \mu^- \mu^+$, forbidden instead in the Standard Model.

^d<https://www.kaggle.com/c/flavours-of-physics/>.

The discovery of such a reaction would therefore be a breakthrough on the laws of nature and a sign of long-sought new physics. Search for those decays performed by LHCb collaboration at that time is described in [24].

3.2. Data description

In this competition, participants were given a list of collision events and their properties. They had to predict whether a $\tau \rightarrow 3\mu$ decay happened in this collision. This $\tau \rightarrow 3\mu$ is currently assumed by scientists not to occur, and the goal of this competition is to discover $\tau \rightarrow 3\mu$ happening more frequently than scientists now can understand.

3.2.1. Signal channel

Different mechanisms can produce tau leptons (mass of τ equals to 1776.86 MeV/ c^2). At LHCb, taus are produced in the decay of heavy flavored particles (containing a c or a b quark), which are listed in Table 1. They are mainly produced in the decays of D_s^- or D^- particles, such as $D^- \rightarrow \tau\eta$. In the simulation samples provided, the correct proportions of the different tau production mechanisms are respected, and the feature `production` identifies the production mechanism.

Table 1. Production mechanisms and their proportions for tau leptons at LHCb, according to the centre-of-mass energy. X_b denotes any particle containing a beauty (b) quark. `production` is a label that denotes the production mechanism of τ for simulated decays. In the data, this label is set to -99.

Mode	7TeV	8TeV	Production
Prompt $D_s^- \rightarrow \tau$	$71.1 \pm 3.0\%$	$72.4 \pm 2.7\%$	1
Prompt $D^- \rightarrow \tau$	$4.1 \pm 0.8\%$	$4.2 \pm 0.7\%$	2
Non-prompt $D_s^- \rightarrow \tau$	$9.0 \pm 2.0\%$	$8.5 \pm 1.7\%$	5
Non-prompt $D^- \rightarrow \tau$	$0.18 \pm 0.04\%$	$0.17 \pm 0.04\%$	6
$X_b \rightarrow \tau$	$15.5 \pm 2.7\%$	$14.7 \pm 2.3\%$	4

3.2.2. Background

The background for $\tau^- \rightarrow \mu^- \mu^- \mu^+$ decay can be divided into two categories. The first one consists of decays with one or more light hadrons (pion or kaon) is wrongly identified as a muon. The main process in this category is $D^+ \rightarrow K^- \pi^+ \pi^+$. The invariant mass distributions for this process are shown in Fig. 3. These two mass distributions differ because of the mass assigned to the final states and thus used when computing the mass of the initial state. On the left-hand side, the muon mass (105.66 MeV) is assigned to all final states, while, on the right-hand side, the correct masses for kaons and pions (139.57 MeV for π^\pm and 493.68 for K^\pm) are used. Hence the misidentification results in a shift in the mass of the initial state.

The second dangerous background $D_s \rightarrow \eta (\rightarrow \mu^+ \mu^- \gamma) \mu^- \nu_\mu$ originates from the decay in which there are three real muons that can mimic the signal signature. This background can be effectively removed requiring all mass combinations of two muons of opposite sign to be greater than 450 MeV.

3.2.3. Additional data

Training of a classifier that is capable of discriminating the signal from the background is a delicate procedure since one can induce unwanted systematic biases that would affect the physics estimations

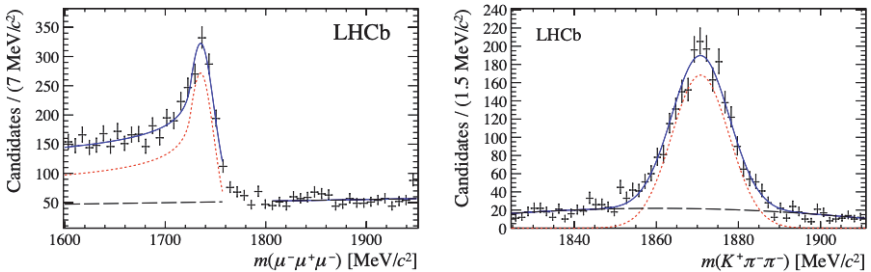


Fig. 3. D meson invariant mass distribution in $D^+ \rightarrow K^- \pi^+ \pi^+$ decays as observed in data. On the left-hand side, all hadrons have been assigned muon mass hypothesis before computing the mass of the mother particle, on the right-hand side, the correct mass hypotheses have been used.

in unpredictable ways. Additional datasets described in the following subsection were supplied to mitigate the risk.

3.3. *Evaluation procedure*

3.3.1. *Verification of the agreement*

Participants have trained classifier models on simulation data for the signal and real data side-bands for the background, so it is possible to reach a high performance by picking features that are not perfectly modeled in the simulation (`min_ANNmuon` is an example of such feature). Organizers demand the classifiers not to have large discrepancy when applied to data and simulation. To estimate the discrepancy a *control channel*, $D_s^+ \rightarrow \phi (\rightarrow \mu^- \mu^+) \pi^+$, is used. It has a similar topology as the signal decay. Organizers provide both data and simulation samples for this decay in the `check_agreement.csv` dataset. The Kolmogorov–Smirnov (KS) test is used to evaluate the differences between the classifier distributions on both datasets. The evaluation constraint was that the KS-value on the test dataset has to be smaller than 0.09.

The cumulative distribution (CDF) functions are computed for simulated data predictions, and real data predictions and Kolmogorov–Smirnov metric is calculated:

$$\text{KS} = \max |F_{\text{simulation}} - F_{\text{real}}|,$$

where $F_{\text{simulation}}$ and F_{real} are cumulative distribution functions for Monte Carlo data and real data, respectively.

3.3.2. *Verification of the correlation*

Correlation of classifier output with the τ mass are unfavorable for data analysis, since those correlations can cause an artificial signal-like mass peak or lead to incorrect background estimations. To prevent cheating, organizers have introduced the Cramer–von Mises (CvM) test [25] to estimate the degree of correlation between the prediction and the mass. The CvM-value of the test has to be smaller than 0.002. Organizers have included the script for

computing the CvM-value, so participants could verify own models on `check_correlation.csv`. CvM metric for the whole dataset predictions CDF is compared to a local (in some mass range) predictions CDF. After that, all intervals are averaged:

$$CvM_{\text{interval}} = \int (F_{\text{global}} - F_{\text{interval}})^2 dF_{\text{global}}, \quad (2)$$

$$CvM = \langle CvM_{\text{interval}} \rangle_{\text{interval}},$$

where F_{global} and F_{interval} are predictions cumulative distribution functions for all the data and data in a given local mass range.

3.3.3. Figure of merit

The calculation of the final figure of merit is performed only if the two above tests are passed on `test.csv` with success and is calculated only using events with $\text{min_ANNmuon} \geq 0.04$.

Initially, the LHCb used the CLS[26] method to determine the upper limit. This method, unfortunately, is computationally expensive and cannot be used in this challenge. Instead, organizers proposed a much simpler metric, which is the weighted area under the ROC curve. The regions and their weights are illustrated by Fig. 4.

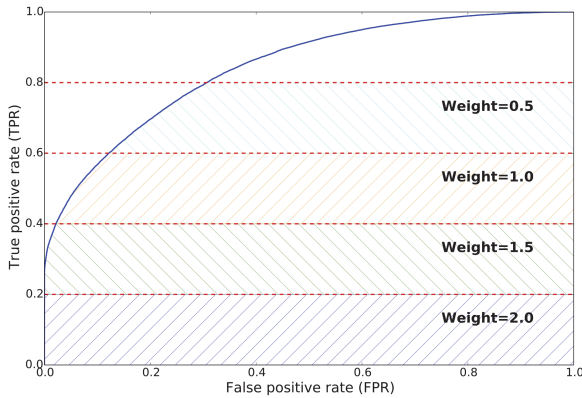


Fig. 4. Weights assigned to the different segments of the ROC curve for the purpose of submission evaluation. The x -axis is the False Positive Rate (FPR), while the y -axis is True Positive Rate (TPR).

The reason to assign different weights to different bins of signal efficiency is that the sensitivity to a given process is not a linear function of expected background events. Most of the sensitivity is obtained when the number of expected background events is $\mathcal{O}(1)$ (see Sec. 2.1 for a different means to achieve the same goal). For example see [27, Table 208].

3.3.4. *Competition datasets*

All the competition data is provided in the following files:

- (1) **training.csv** is a labeled dataset (the **signal** being 1 for signal events, 0 for background events) to use for training the classifier. Background events come from real data mass side-bands and from the simulation.
- (2) **check_agreement.csv** is a labelled dataset (the **signal** being 1 for simulated data, 0 for real data) with the same features as in the **training.csv**. This dataset is used to check the agreement between simulated and real data as described in Sec. 3.3.1.
- (3) **check_correlation.csv** is a dataset with the same features as the **training.csv**, to check correlation of the classifier with the τ mass as described in Sec. 3.3.2 before submission.
- (4) **test.csv** is a non-labeled (signal and background are mixed) dataset, containing (a) simulated signal events and real background data, (b) simulated events and real data for the control channel.

The setup for the challenge was unusually complicated. The correlation and agreement checks was introduced specifically to match the intuition of physics checks with Kaggle platform requirements. So the organizers have prepared special kind of prizes to the community to mitigate the risk that smart participants find a way to bypass those checks.

3.4. *Prizes and participation statistics*

The competition was running for three months and has attracted 673 teams. Participants submitted more than 10 thousand different solutions. The main prize allocation for the competitors was:

USD 15,000 as judged by official Kaggle leaderboard. Additionally, a special Physics Prize that was awarded to teams that, as judged by the LHCb collaboration members, create a model that is most useful from the physical perspective. The main motivation for the special prize was that the challenge setup was quite tricky for regular Kaggle challenge. So it could provide an incentive for the participants to find workarounds to bypass the checks and produce meaningless solutions from the Physics perspective. Figure 5 shows the private leaderboard statistics. One can see that the top teams have reached the ideal score of 1.0, which turned out to be a clever way to bypass additional checks while still introducing unwanted selection properties. Indeed, the trick of exponentiation^e allowed to get a rather high score in the leaderboard. Organizers have scrutinized solutions by

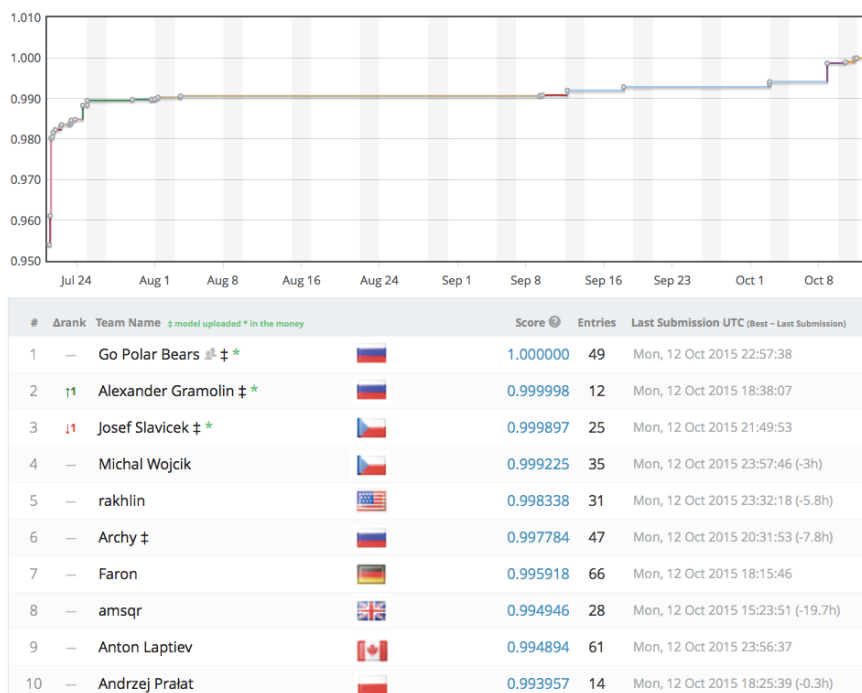


Fig. 5. Top private “Flavor of Physics” leaderboard score evolution.

^e<https://www.kaggle.com/rakhlin/abcde/code>.

top 20 participants and have identified several solutions that avoided unphysical score overfitting (see Sec. 3.5).

3.5. *Physics prize and follow-up workshop*

The Heavy Flavor Data Mining workshop was organized at the University of Zurich in February 2016^f to wrap-up the results of the competition and to award the physics prizes. The recipients were Vicens Gaitan and Alexander Rakhlin. The main ideas of their approaches are highlighted below.

3.5.1. *Data doping by Vicens Gaitan*

The idea is to “dope” (in the semiconductor meaning) the training set with a small number of Monte Carlo events from the control channel but labeled as background. It disallows the classifier to pick features discriminating data and Monte Carlo. Figure 6 illustrates the doping procedure.

Such a procedure involves two parameters that regularize the learning: (a) The number of “doping” events and (b) the complexity of the classifier (for instance number of trees). Those can be tuned depending on the problem and data at hand.

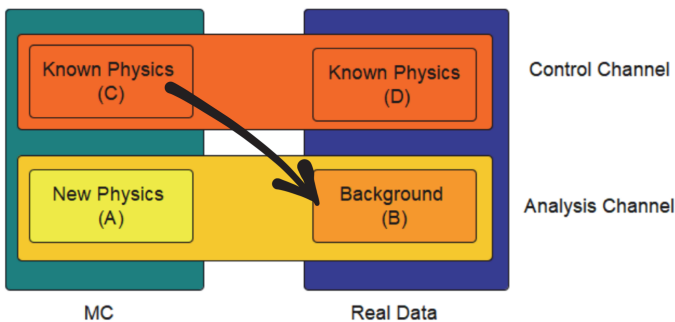


Fig. 6. Data doping: the training set with a small number of Monte Carlo events from the control channel, but labeled as background.

^f<https://indico.cern.ch/event/433556/>.

3.5.2. *Transfer learning by Alexander Rakhlin*

The network is trained in the two-stage process.

- (1) create a strong model for the signal channel using all available features. This model is based on an ensemble of 20 feed-forward fully connected neural nets.
- (2) transfer this model to control channel. The model's output is stacked with original features and cascaded to additional “*transductive*” neural network of similar configuration. The purpose of the second net is to track the first model's output on the signal channel with minimal and controlled adaptation to control channel.

All three metrics (AUC, KS, CVM) are global and not analytically differentiable. It makes gradient descent generally impossible. So the procedure is the following. Initial weights of the transductive network are set to reproduce the output of the original model; this is accomplished after 1–3 epochs of standard GD with cross-entropy loss on the signal channel. Adaptation of weights is made with stochastic optimizer using Powell's method. Loss function incorporates AUC, KS, CVM metrics and allows controlling them during optimization. As a result, it obtains best of the two worlds: performance on signal channel preserved as much as possible (slightly drops only in the 3rd decimal place), the tests on control channel passed. To keep the model as physically sound, one can control its performance on the signal channel during optimization. Furthermore, it is possible to restrict the transductive network from excessive deviation from its original state (weights) or implement any other regularizer.^g

3.6. *Conclusion*

The Flavors of Physics challenge was aiming at an ambitious goal of finding a way to deal with nuisance parameters and MC/real data

^g<https://github.com/alexander-rakhlin/flavours-of-physics>.

discrepancies happening in particle physics analyses. In real-life settings, all the checks are performed by professionals with physical intuition that helps them to keep solutions under meaningful constraints. Translation of those constraints to the competition platform implies taking the risks of (a) simplifying of the limitations and (b) giving incentive to the participants to hack the metric. It has happened to the challenge, and one can follow the details on Kaggle forum. Nevertheless, special prizes allocated by the organizers, which had to be awarded by the domain scientist committee has allowed motivating development of physically-sound solutions that were outlined above.

4. TrackML

4.1. Introduction

The Tracking Machine Learning (TrackML) challenge^h took place mainly in two phases, an Accuracy phase [28] in 2018 on the Kaggle platform,ⁱ and a Throughput phase [29] (combining accuracy and speed) in 2018–2019 on Codalab,^j preceded by a limited scope 2D prototype competition in April 2017 [30]. The challenge is described in details in the papers referenced above, only a summary is given here, focusing more on the lessons (see also [31]).

The analysis pipelines of the proton collisions at the Large Hadron Collider (or *events*) rely on a first step, the reconstruction of the trajectories of the particles within the innermost parts of the detector. The time to reconstruct the trajectories — in a constant magnetic field these would follow a helical path — from the measurements (3D points) is expected to increase faster than the projected computing resources. New approaches to pattern recognition are thus searched for to exploit fully the discovery potential of the High Luminosity Large Hadron Collider. A typical event for ATLAS or CMS detector at HL-LHC design luminosity would have about 100,000 points

^h<https://sites.google.com/site/trackmlparticle/>.

ⁱ<https://www.kaggle.com>.

^j<https://competitions.codalab.org>.

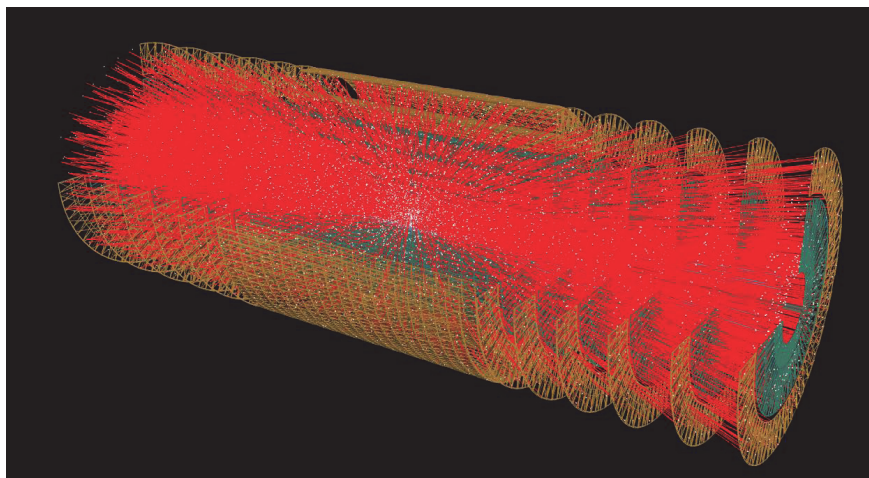


Fig. 7. TrackML detector (one sector of the detector has been etched out). White dots are the measured points, while the red lines are the trajectories of the particles (from [28]).

to be associated into 10,000 trajectories (see Fig. 7). The state of the art was about 10s per event on a modern CPU when the challenge was designed. Given that 10–100 billions such collisions need to be treated each year, the importance of a significant increase of the reconstruction throughput becomes evident.

The goal of the TrackML challenge was to expose the problem of fast particle tracking to the wide Computing Science community. Since designing new algorithms and writing fast software require separate expertise, it was decided early to split the competition into two phases: the first phase (Accuracy) would only be about algorithm accuracy, while the second (Throughput) would be about fast software with a good compromise on accuracy.

While for the Accuracy phase, participants had to upload a solution file to Kaggle platform indicating how the points are clustered into tracks, for the Throughput phase participants had to upload their software directly to the Codalab platform, on which it was executed in a controlled environment. By doing so, the resource usage was measured in a standardized way, and the Throughput score was then derived from the accuracy and the speed.

4.2. Dataset and score

A dataset consisting of an accurate simulation [32] of an LHC-like experiment has been created, listing for each event the measured 3D points, and the list of 3D points associated to a true track.

The detector simulated is a full Silicon detector organized in cylinders and disks sharing the same axis of symmetry z ; the origin of the axis is at the centre of symmetry. An approximately solenoidal magnetic field of the same axis bends the particles so that their trajectories are approximate arc of helices. Most, but not all particles, start from close to the origin.

The participants to the challenge should find the tracks, meaning building the list of 3D points belonging to each track, in an additional test dataset without the ground truth.

Detailed algorithm performance studies usually involve in-depth analysis of hundreds of histograms. Yet, as usual for a competition, algorithms should be ranked from a single score number to be maximized. The Accuracy score was chosen to be “the weighted fraction of points correctly assigned”, which is computed from the point association inferred by the participants. This choice for an Accuracy score based on points was somewhat counter-intuitive, as it is much more usual in the community to evaluate the performances in term of found tracks, examining the tracking efficiency (fraction of tracks found) and quality (precision of the reconstructed track parameters). The post-competition in-depth analysis of the algorithms submitted has shown that this choice has been correct, as the best algorithms in terms of the score also had the best performances when analyzed in depth. It should be noted that this score is non-standard and required specific development by the Kaggle Data Scientist in charge of the competition.

For the Throughput competition, the simulation has been slightly adjusted. An ad-hoc score combining the Accuracy and the speed has been devised. The iso-score lines appear, Fig. 8, pushing participants to arrive closest to the bottom right corner, with the largest Accuracy score and smallest per-event time. Participants got a non-zero score only if their submission could achieve more than 50% accuracy

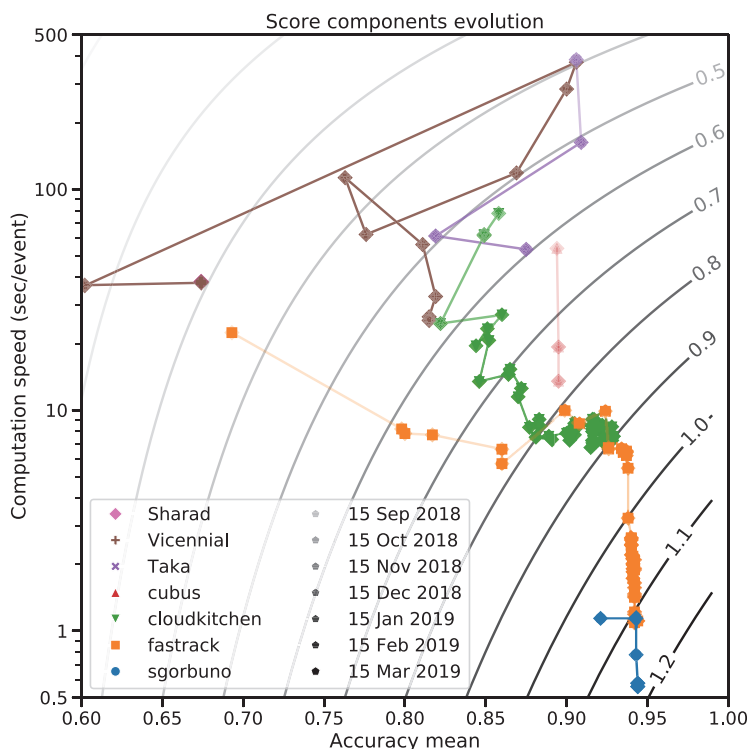


Fig. 8. TrackML Throughput-phase participants score evolution. The horizontal axis is the mean accuracy over the 50 test events, and the vertical axis is the average computation speed per event. The total score, function of both variables, is displayed in gray contours. Each color/marker type corresponds to a contributor, the lines help to follow the score evolution (from [29]).

(to avoid poor but super-fast algorithm) in less than 600 s per event (to avoid straining the resources dedicated to the challenge).

The Throughput evaluation required a reliable measurement of the inference time on well-defined resources, which was not possible (at least at the time) on Kaggle. It was then chosen to have the Throughput phase on Codalab which offers this possibility. It meant developing the code which would run the submitted software in a Docker environment with resources limited to 2 CPU and 4 GB of total memory, and the code evaluating the Accuracy score (for the Accuracy phase this was done by the Kaggle Data Scientist relying for

a large part on existing internal Kaggle code). The participant code is embedded in a skeleton taking care in particular of the reading of the event data and writing out the solution so that the time measured is purely the one of the inference. Several issues were uncovered and solved:

- the time measurement was found to be reproducible only within 2%, which could have lead to a change of ranks in case of many participants. Hence it was decided to measure the time after the end of the competition by averaging multiple (10) runs on a new dataset. This was done after the end of the competition and the updated time measurements were very close to the one provided online.
- it could have been the case that participants write in the log file useful information about the test dataset, and then use it in a later submission. To avoid this, logging was completely disabled.
- it was expected (but not enforced) that participants would submit their source code which would be compiled on the platform. Uploading additional libraries was allowed given it was not expected the Docker environment to be complete with all possible utilities. However, some participants chose to upload their code directly as a library, which prevented the organizers to see their code during the competition.
- all sophisticated hacks could not be excluded; instead of multiple safety measures, for which the organizers had no time nor expertise, hacking was forbidden in the rules of the challenge, and the submission of the software required to win any price was expected to deter hacking effectively. No sign of hacking was detected after the competition.

4.3. Competitions

The TrackML Accuracy phase has run on Kaggle 1st May 2018 to 10th August 2018. The TrackML Throughput competition opened a few weeks later, the 3rd September 2018. It was initially due to end 18th October 2018, but given the lack of competitors, it was extended till 15th March 2019.

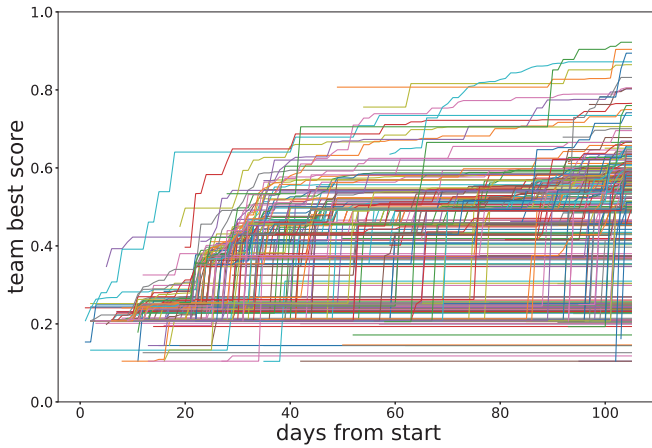


Fig. 9. Evolution of the best score of each team as a function of time (from [29]).

The Accuracy phase was well attended, with a total of 656 participants. Figure 9 shows the evolution of the leader scores throughout the competition. There is initially a large cluster of candidates achieving a score of 20% to 25%, which corresponds to the 22% performance of the DBSCAN starting kit. After around 30 days, public kernels achieving a performance greater than 50% (still based on DBSCAN) were posted on the public forum by some participants, which leads to a second group of candidates reaching a performance of 50% to 60% after 40 days of competition. Finally, a score of more than 90% was only reached in the last days of the competition. Front runners are well separated from the pack and from each other, which is a clear indication of the complexity of the competition (if this had been a Tour de France stage, it would have been a mountain stage rather than a peloton finish in a flat stage).

The competition forum has been very active with participants posting visualization notebooks and algorithm kernels. The accompanying documentation provided minimal information on existing HEP tracking algorithms in order to not bias the competition towards existing solutions. However, participants know how to google and have searched and found and posted in the forum technical papers on tracking, courses and even the Ph.D. dissertation of one of the organizer.

The post-competition analysis has uncovered that a variety of approaches was used, among which:

- DBScan is a popular (in ML) clustering algorithm, to build clusters of nearby points in a space of n -dimension. The points belonging to a track are on an arc of helix, they are not close in 3D geometrical space. However, they can be brought to be close after suitable transformations. A simple example was provided to all participants as a starting kit, which allowed to give non-random results with a few lines of a code. Somewhat to the surprise of the organizers, the algorithm was further refined by many participants and all the way up to rank #9 by CPMP who was given one jury prize.
- Hough transform, a classical (in HEP) algorithm mapping the geometric space to the track parameters space where the clustering is done was used by some participants and brought to rank #7 by Yuval and Trian using several tricks to make it work in this situation.
- More innovative, the **finnies** have used Recurrent Neural Network to do the track following reaching rank #12.
- The most astonishing algorithm was from **outrunner** rank #2 with a combination of a Neural Network and brute force: it first trains a NN to regress the probability that any pair of points belong to the same track. Then at the inference stage, it builds the large $100k \times 100k$ matrix with the probability of all possible pairs of point. And finally, it builds the tracks by picking one by one the most likely pairs. It does work but is very compute-intensive, about one full day per event, which makes it unpractical.
- The post-challenge performance analysis has revealed somewhat accidentally that **diogo**, rank #100 was the only one able to reach high efficiency for rare abnormal tracks coming far from the origin. This was achieved with an algorithm keeping track of connection between large voxels. It is unpractical as soon as there is some density of similar tracks but it is quite interesting for abnormal track finding.
- The classical (in HEP) track following algorithm has been improved with non-classical techniques by top performers, in particular **Top Quarks** rank #1, **Sergey Gorbunov** rank #3,

`demelian` rank #4, they will play an important role in the Throughput phase detailed now.

It was hoped that many participants (or new ones) will carry on in the Throughput phase. This did not really happen, despite the extension of the deadline from October 2018 to March 2019. Only seven participants got a non-zero score. In hindsight, this can be understood to come from a combination of factors:

- the lesser popularity of Codalab compared to Kaggle, where people can earn points across competitions;
- the complexity of the problem;
- the necessity to write C++ code, when a typical Kaggle participant is more used to python;
- given the threshold at less than 600 s per event and more than 50% efficiency, it was already non-trivial to have a non-zero score;
- despite all the efforts to document and streamline the procedure to submit a solution, it still required a larger commitment than for a typical Kaggle competition. Also, the fact that log files were hidden to participants made debugging more difficult for them.

Nevertheless, the small number of participants has been more than compensated by the high quality of the top three participants (see Fig. 8), who have all reached above 90% accuracy with a time up to 0.5 s, while the original goal was around 10 s per event.

The original idea was that algorithms developed in the Accuracy phase would be optimized and adapted to the second phase, possibly not by the same participants. This was not enforced in any way but in fact, it happened:

- Sergey Gorbunov (pseudo `sgorbunov`) rank 1 in Throughput phase had obtained rank 3 in the Accuracy phase;
- Dmitry Emliyanov (pseudo `fastrack`) rank 2 in the Throughput phase had obtained rank 4 in the Accuracy phase (with pseudo `demelian`);
- Marcel Kunze (pseudo `cloudkitchen` rank 3 in Throughput phase) used as a starting point the algorithm of `TopQuark`, rank 1 in the Accuracy phase, and has largely augmented it.

4.4. *Scientific conclusion*

It is not possible to compare directly to in-house algorithms which would need to be adapted to this specific dataset. Also, they usually ignore the numerous tracks with p_T less than 400 MeV (the tracks with the highest curvature) while algorithms presented here can reconstruct tracks down to 150 MeV. It can be estimated that in-house algorithms are not faster than 10 s per event on one CPU core, so one order of magnitude slower than Mikado from Sergey Gorbunov (a.k.a. `sgorbuno`), 0.5s on two CPU cores. On the other hand, several simplifications were done in the dataset (in particular neglecting the sharing of points between tracks) so that it remains to be seen whether the new algorithms can live up to expectations when used in the ATLAS and CMS experiment context. The community is now in the process of doing this exercise.

In the end, Machine Learning was not at the core of the three best Throughput algorithms. Nevertheless, after extended discussions between the three winners and experts in the field, a consensus appears that there are two likely avenues for the use of Machine Learning in such problems (i) combine ML with classical discrete optimization, for example using a classifier to select early and quickly the best candidates as done by Marcel Kunze a.k.a. `cloudkitchen` (ii) use ML to automatize the lengthy tuning of the internal parameters of the algorithms (circa 10,000 in the case of Mikado by Sergey Gorbunov).

4.5. *Organization conclusion*

The organization of the TrackML challenge was a long process, the main elements of the timeline are indicated below:

- March 2015 Berkeley Initial discussion Connecting The Dots workshop;^k
- March 2016 Vienna More discussion Connecting The Dots workshop,^l team is being set up;

^k<https://indico.physics.lbl.gov/event/149/>.

^l<https://indico.hephy.oeaw.ac.at/event/86/>.

- April 2017 2D hackathon Orsay Connecting The Dots workshop.^m Follow up paper released end 2017 [30];
- May 2017 first contacts with Kaggle;
- March 2018 Connecting The Dots workshop Seattleⁿ 3D hackathon, feedback on almost final dataset and score;
- May–August 2018 Accuracy challenge on Kaggle,^o follow-up paper released early 2019 [28];
- December 2018: NeurIPS competition workshop, with top participants invited;
- Oct 2018–Mar 2019: Throughput challenge on Codalab;^p
- July 2019: CERN Grand finale workshop^q with top participants invited;
- October 2019: Université Paris-Saclay Institut Pascal Advanced Pattern Recognition workshop^r with top participants invited for two weeks;
- May 2021: final paper release [29].

Also, there were more than 40 presentations at physics conferences (ICHEP, CHEP, EPS, etc.), Machine Learning conferences (WCCI, NeurIPS CiML and Competition workshops), seminars in physics departments, python meetup (Paris, Genève) by all members of the team.

As can be seen, as the challenge develops, several events were organized to get feedback from the wide tracking experts community. These were important to exercise and adjust the mechanics of the challenge and discuss the conclusion and long term impact.

In particular, after the first round of initial discussions, a prototype has been the organization of a challenge [30] on the RAMP^s platform during the Connecting The Dots workshop^t (a workshop

^m<https://ctdwit2017.lal.in2p3.fr>.

ⁿ<https://indico.cern.ch/event/658267/>.

^o<https://www.kaggle.com/c/trackml-particle-identification>.

^p<https://competitions.codalab.org/competitions/20112>.

^q<https://indico.cern.ch/event/813759/>.

^r<https://indico.cern.ch/event/847626>.

^s<https://paris-saclay-cds.github.io/ramp-docs/>.

^t<https://ctdwit2017.lal.in2p3.fr>.

for experts in pattern recognition) held at IJCLab in Orsay in March 2017. The problem was essentially the same as the one exposed here but very much simplified to be a 2D problem with just 20 tracks per event (instead of 10,000 in 3D). There was no speed constraint. The same accuracy score was used for the first time. This 2D challenge has already yielded a variety of algorithms (not directly applicable in 3D though) and demonstrated that the accuracy score was indeed selecting the best algorithms. Its success set a green light to launch the full project.

The team comprised 19 people, senior scientists with expertise in the field of tracking or Machine Learning, post-docs and students, all of them part-time. The total effort can be estimated to be 3 Full-Time Equivalent year. The main tasks were: preparing the dataset, the accompanying documentation, helper library and starter kit for the 4 hackathons and competitions organized, interacting with Kaggle, implementing the competition in Codalab, searching for sponsors, running the competition, organizing the different associated workshops, doing the post-competition analysis and writing the papers.

Sponsoring was needed for the prizes (\$30k and 1 NVidia V100 for the Accuracy phase, 15k euros and 1 NVidia V100 for the Throughput phase) and for the invitations to NeurIPS 2018 Competition workshop (Accuracy phase) and CERN July 2019 (Throughput workshop).

4.6. *Follow-up studies*

Separately, the availability of the TrackML dataset has been extremely useful to facilitate the collaboration of experts which are usually working on their own data within their own experimental team. It has been used for new studies like investigating tracking with simulated annealing on a D-Wave quantum computer [33], or with graph networks [34] (see also Chapter 12 in this book). Somewhat unexpectedly, the dataset has also been used to explore the usage of Augmented Reality to visualize scientific data [35, 36]. The datasets have been released on Zenodo [47, 48].

5. LHC Olympics

5.1. Introduction

Despite an impressive and extensive effort by the Large Hadron Collider (LHC) collaborations, there is currently no convincing evidence for new particles produced in high-energy collisions. LHC Olympics 2020 Anomaly Detection Challenge challenge was aimed at exploring the capabilities of machine learning to enhance the potential signal of Beyond Standard Models (BSM) using all of the available information.

5.1.1. The challenge goal

The challenge goal was to ensure that the LHC search program is sufficiently well-rounded to capture “all” rare and complex signals. Different stages of the competition are focused on different volumes of the phase space since potential BSM parameter space is vast.

5.1.2. Challenge setup

The LHC Olympics 2020 setup is aligned with the first LHC Olympics organized in 2005–2006.^u Participants are provided with two types of data:

- “Monte Carlo Simulation Background”: This is a simulated sample that does not have a signal. Be warned that both the physics and the detector modeling for this simulation may not exactly reflect the “Data”.
- “Data”: These samples contain a mixture of background with some new signal(s). Three unique samples referred-to as *black boxes* have been released during LHCO 2020 challenge. All the samples had become available in November 2019. The first sample has been unveiled mid-January, during winter part of the LHC Olympics, and the remaining two has been unveiled during the LHC Olympics summer workshop.

^u<https://public-archive.web.cern.ch/en/Spotlight/SpotlightOlympics-en.html> and <https://www.kitp.ucsb.edu/activities/lhco-c06>.

Both the “Simulation” and “Data” have the same event selection criteria (see Sec. 5.2). Participants had to find signals of BSM in “Data” samples and to report various metrics that estimate the confidence of those findings. There were two workshops during 2020 focused on the discussion of multiple techniques and intermediate challenge results. The organizational committee of the LHC Olympics 2020 coincides with the one of those workshops.^v

5.2. Data description

For both background and black box data, events supposed to have the form of $X \rightarrow \text{hadrons}$, where X is a new massive particle with an $\mathcal{O}(\text{TeV})$ mass. Events are selected after a single trigger of anti- k_t $R = 1.0$ jet [37] with pseudorapidity $|\eta| < 2.5$ and the transverse momentum $p_t > 1.2$ TeV. Number of events per data sample is the same and equals to 1M.

These events are stored as pandas DataFrames saved to compressed HDF5 format. For each event, all reconstructed particles are assumed to be massless and are recorded in detector coordinates (p_t, η, ϕ) . More detailed information, such as particle charge or type, is not included. Events are zero-padded to constant size arrays of 700 particles. The array format is therefore (1M, 2100).

5.2.1. Background

The background sample of 1M events consists of QCD dijet events simulated using Pythia8 and Delphes 3.4.1. Both the physics and the detector modeling for this simulation are not guaranteed to precisely reflect the signal “data”.

5.2.2. Signal

The signal dataset is split into three files, referred to *black boxes*. Each “black box” contains 1M events meant to be representative of actual LHC data. These events may include BSM signal(s), i.e. it contains either mixture of some signal and background or just background.

^vGregor Kasieczka, Benjamin Nachman and David Shih.

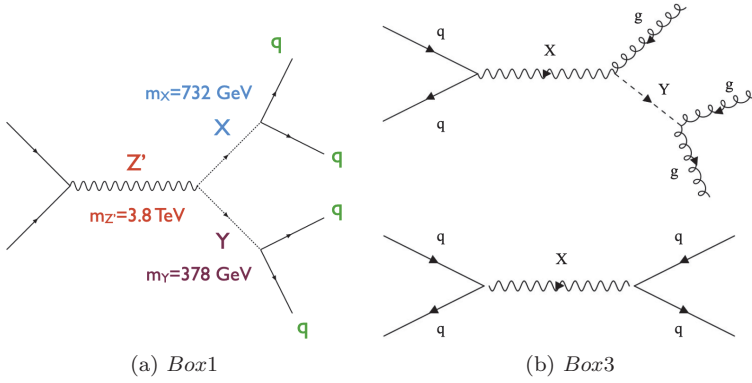


Fig. 10. The new-physics modes hidden in the black boxes.

The signal in the former case represents a kind of new physics simulated by the same software packages as the background. Figure 10(a) represents the process hidden in the first box. There were 834 events of this kind. The second black box did not contain any signal and was filled with the same QCD background events to check the participant's algorithm false positive rate at the boundaries of the phase space. The most complicated case was hidden in the third box. It required to stack together two decay modes depicted at Fig. 10(b) with $m_X = 4.2$ TeV, $m_Y = 2.2$ TeV, $BR(X \rightarrow qq) = 1 - BR(X \rightarrow Yg) = 0.375$. Work [38] inspires this physics, i.e. simple extensions of RS motivated by LHC Run I null results and little hierarchy problem, where X represents Kaluza–Klein gluon and Y–IR radion. The total number of signals in the third box was equal to 3200.

The competition datasets are published at Zenodo archive [39].

5.3. Evaluation procedure

Participants should report:

- a p -value associated with the dataset having no new particles (null hypothesis);
- a description of the new physics, as complete as possible. For example the masses and decay modes of all new particles (and uncertainties on those parameters);

- number of signal events (+uncertainty) in the dataset (before any selection criteria).

Outcomes will be judged based on (a) the optimality of the p -values and (b) the accuracy of the new physics characterization:

- optimality corresponds to the “best” p -value will be the lowest reported p -value that is above the fully optimal p -value (as determined with a fully supervised deep learning classifier);
- accuracy is computed by the number of sigmas from the right answer is used wherever applicable. Number of sigmas is estimated as $|(\text{predicted} - \text{true})/\text{predicted_uncertainty}|$.

Organizers prepared the competition starting kit^w with scripts that read in the data and perform exploratory data analysis with it.

5.4. *Prizes and participation statistics*

There was no money prize associated with the competition and, perhaps, it was mainly meant for particle physicists since the entry required some understanding of basic QCD models and new physics models. Thus, it has attracted a couple of dozens of participants during 2020 mostly from physics departments. It was also relatively lightweight in terms of evaluation tools; competitors had to submit all the metrics via a google form. Such form allowed to collected extended feedback like a description of the new physics a participant was aiming for. The organizers put together two workshops during the winter^x and the summer^y of 2020. Every workshop has many relevant contributions to the inclusive search for the new physics. Detailed workshop outcome analysis is available as a workshop contributions.^z

^w<https://github.com/lhcolympics2020/parsingscripts>.

^x<https://indico.cern.ch/event/809820>.

^y<https://indico.desy.de/event/25341>.

^z<https://indico.cern.ch/event/809820/contributions/3708303/attachments/1971116/3347225/SummaryTalk.pdf>, <https://indico.desy.de/event/25341/contributions/56822/attachments/36777/45997/SummaryAnomalyDetectionWorkshopJuly2020.pdf>.

5.5. Conclusion

The LHC Olympics presents a charming and successful format for running a competition in sustainable and cost-saving mode. It is more focused on the physic-oriented results rather than attracting a broader data-science audience. The main page of the challenge^{aa} includes references to several papers describing participant's contributions. Also, there is a community whitepaper on the competition outcomes scheduled to be published. It welcomes every participant for the co-authorship. LHC Olympics organizers invite new BSM black boxes from the community for the future runs of the challenge.

6. Competitions Platforms

6.1. Platforms for data challenges

There is a dozen of platforms that allow hosting data challenges. These differ significantly in terms of flexibility, functionality and community factors. As it was mentioned in the previous sections, challenges requirements can be quite diverse and demanding. This section gives an overview of the leading players and alternative approaches that can be used to run a new challenge. A platform for hosting a data challenge is a service that is provided by a company or institution behind it. Those services usually follow the so-called Common Task Framework (CTF) (see Sec. 1 and paper [2] for details).

Historically different groups started to develop such services around different challenges; thus, its functionality may differ. Anyway, since those follow the common competition protocol, every platform allows to upload a dataset, describe challenge condition, setup evaluation procedure and invite the community to a new challenge. However, some platforms are better for dealing with human-in-the-loop evaluations; some give better flexibility in terms of metric specification; some can deal with private data. We are going to overview the features of the following platforms:

^{aa}<https://lhco2020.github.io/homepage/>.

- AICrowd^{bb} by EPFL, AICrowd,
- CodaLab^{cc} by Chalearn and Université Paris-Saclay,
- CrowdAnalytiX^{dd} by CrowdAnalytiX,
- EvalAI^{ee} by CloudCV,
- Kaggle^{ff} by Alphabet Inc,
- RAMP^{gg} by Université Paris-Saclay Center for Data Science,
- Tianchi^{hh} by Alibaba,
- TopCoderⁱⁱ by TopCoder.

The list is not meant to be comprehensive, as it is focused on active platforms with broad communities as of the end of 2020, i.e. with more than ten competitions started during 2020 using publicly available information. Also, there are platforms like Grand Challenge^{jj} that are focused on some narrow scientific domain. We have identified the following criteria for comparing the platforms above that are relevant for HEP-related competitions.

6.1.1. *Criteria*

We outline the main characteristics that we use for the comparison. Those criteria are sorted by order of relevance to the competitions described in this book chapter.

Code sharing, reproducibility: challenge participants are not always motivated by getting the highest score. Instead, they might want to explore new things or to get praised by the community. Thus, the ability to share and discuss their code with other participants becomes a crucial feature for new complicated challenges like the

^{bb}<https://www.aicrowd.com/>.

^{cc}<https://competitions.codalab.org/>.

^{dd}<https://www.crowdanalytix.com/>.

^{ee}<https://eval.ai/>.

^{ff}<https://www.kaggle.com/competitions>.

^{gg}<https://ramp.studio/>.

^{hh}<https://tianchi.aliyun.com/competition/>.

ⁱⁱ[https://www.topcoder.com/challenges?tracks\[DS\]=true](https://www.topcoder.com/challenges?tracks[DS]=true).

^{jj}<https://grand-challenge.org/>.

HEP ones. Sometimes code-sharing is available right within the platform, like at Kaggle, or sometimes participants can link their solution to github repository/commit, so other participants can reproduce and play with it. Such feature adds greatly to the reproducibility of the winning solutions making it much more scientific.

Code submission: accuracy metrics are not enough to evaluate the dynamic aspects of a participant's code. In some cases like tracking or triggering, one may be interested in comparing the accuracy of an algorithm only if certain execution speed/resource consumption constraints are met. Thus, some platforms support code as a submission to evaluate a solution to the full extent.

Community activity: it is a cumulative estimation that takes into account the total number of challenges organized, number of challenges in 2020, the maximum number of participants per challenge and estimated size of the community.

Custom metrics: the ability to implement custom metrics is crucial for some non-trivial cases that wish to compare algorithm performance for non-usual challenges like it was for Flavor of Physics. Sometimes it is needed to make a trade-off between accuracy and performance like it was for TrackML. Some platforms allow choosing just one among many predefined metrics; some allow for custom implementations. Some platforms charge an additional cost for implementing non-standard metric.

Staged challenges: sometimes challenges might look too weird at the beginning, so it helps to split it in smaller chunks. Thus, it is possible to mitigate risks of data leakage by adding a preliminary stage and testing the competition settings. It will help people to keep the context between stages and smoothly transfer knowledge of the best solutions.

Private challenge evaluation: data privacy is a serious issue even in fundamental science, so some platforms allow running participant's solutions evaluation using an organizer's dedicated machines, thus one setup a challenge without the need to share restricted datasets.

Open-source: in usual scenarios, a platform operates as a service and challenge organizers do not care much about tweaking its functionality. However, open-source gives the ability to evaluate the project activity, check the details of platform evaluation mechanics or run own instance of the service for some local events with private datasets, for example.

Human-evaluation: some challenges do not have ground-truth labels in the data. For example, evaluation of a dialog bot requires communication with a living person, or images of galaxies labeled by an agent may require extra human validation. Some platforms allow connecting to human-evaluation platforms such as Amazon Mechanical Turk (see below) or alike.

Reinforcement-Learning (RL) evaluation: agents designed to operate in a dedicated environment present another challenging task for a fair evaluation for a couple of reasons: (a) environments can be very diverse, (b) each agent may require considerable computational resources the platform needs to account for, (c) each agent operates in randomized environments thus it may require several evaluations to get a statistically-sound score.

Run for free: many platforms allow setting up money prizes to the competition winners, so they charge for running those settings. Some platforms still allow to run a challenge almost for free — if the problem is not computationally heavy, it is possible to run it using the service infrastructure. Sometimes it is possible to connect the organizer’s computational resources to the service, thus avoiding extra charges.

6.2. Comparison

Overview of the platforms concerning the criteria above is presented in Table 2. In addition to the overall comparison, it is worth mentioning individual features that are difficult to fit into a generic table.

RAMP: Rapid Analytics & Model Prototyping is a service that is mainly used by the Université Paris-Saclay Center for Data Science to support own events like hackathons or datacamps. Remarkably,

Table 2. Platform overview.

Criteria	AICrowd	CodaLab	CrowdAnalytiX	EvalAI	Kaggle	RAMP	Tianchi	TopCoder
Code-sharing	✗	✓	✗	✓	✓	✓	✓	✗
Code submission	✓	✓	✗	✓	✓	✓	✓	✓
Active community	★★	★★★★	★	★★	★★★★★	★	★★★★	★★
Custom metrics	✓	✓	✓	✓	✓	✓	?	✗
Staged challenge	✓	✓	✗	✓	✗	✓	✗	✗
Private evaluation	✗	✓	✗	✓	✗	✗	✗	✗
Open-source	✓	✓	✗	✓	✗	✓	✗	✗
Human evaluation	✗	✗	✗	✓	✗	✗	✗	✗
RL-friendly	✓	✗	✗	✓	✗	✗	✗	✗
Run for free	✗	✓	✗	✓	✓	✗	?	✗

RAMP involves two phases of each event — competition and collaboration. During competition phase participants, try to design their algorithms, while upon collaboration stage, they share their approaches and team up for the sake of a better solution. RAMP is published under BSD-3 license,^{kk} so it may come handy for a lightweight setup of an event at own premises.

Kaggle: allows to run a competition entirely free for non-commercial purposes in so-called *InClass* mode: (a) so Kaggle does not advertise it to the community and (b) gives a limited setup flexibility, i.e. one cannot evaluate submissions against anything but the set of pre-defined straightforward metrics like RMSE or ROC AUC. Also, such competitions do not award any Kaggle ranking points to the participants, which reduces the incentive to join it significantly. Once a company/university decides to run a full-fledged public challenge, it is possible to implement a custom metric, but it may turn out to be quite expensive.

Tianchi: despite a relatively young age, it is a top-rated service in China with very similar to Kaggle functionality that includes running kernels and ranking points. Challenges quickly can gain several thousand participants. However, most of the audience is Chinese, so communication skills in Chinese would come handy.

TopCoder: is one of the oldest and biggest worldwide platforms for outsourcing coding tasks. Thus the audience is huge — 1.5 million of users. However, it added machine learning tasks in 2018, so only a fraction of the total users is relevant for addressing data-driven challenges.

6.3. *Alternative approaches*

The platforms from the comparison above implement challenges along Common Task Framework [2]. However, it is not the only option. Below is a list of platforms that rely on different assumptions and implement peculiar interaction protocols.

^{kk}<https://github.com/paris-saclay-cds/ramp-board/>.

Amazon Mechanical Turk (AMT)¹¹: is a marketplace for completion of virtual tasks that require human intelligence. A business or academics typically use it to label data that later on can be used for training ML algorithms. AMT has been around for more than 15 years. Major companies like Google and Microsoft have similar versions of such marketplaces.

Zooniverse^{mm}: While AMT focuses on pretty generic tasks like reading labels from images, captcha translation, listening comprehension, tagging inappropriate images, etc. Zooniverse builds a community of people that are interested in contributing their efforts and intelligence to scientific research advances. It provides participants with unlabeled datasets from a wide variety of scientific branches: biology, climate, history, physics, etc. Those datasets require human intelligence not only for labeling but also for understanding the scientific assumptions of the domain and phenomena presented. Participation in real-science research can motivate people quite significantly. There are cases when discussions between scientists and Zooniverse participants lead to new scientific discoveries [40].

OpenMLⁿⁿ: is an online machine learning platform for sharing and organizing data, machine learning algorithms and experiments. Founders of the platform are passionate about the comparison of different ML methods. Thus they have created the service that allows to run an algorithm across different datasets and systematically compare its performance. While there are no private leaderboards, every check is performed via system API and protocol systematically. Thus new experiments are immediately compared to state of the art without always having to rerun other people's experiments. The recent development of OpenML involves the design of AutoML evaluation framework for a broad spectrum of datasets.

PapersWithCode (PwC)^{oo}: organizes access to technical papers that also provide the software used to create the paper's findings,

¹¹<https://www.mturk.com/>.

^{mm}<https://www.zooniverse.org/>.

ⁿⁿ<https://www.openml.org/>.

^{oo}<https://paperswithcode.com/>.

has grown immensely in the past few years. With the help of this platform, one can find the most current state of the art to the problem of the interest and read details of the method in the linked paper from arXiv.

InnoCentive^{PP}: is an innovative hub for a new kind of problem-solving. It describes the framework of “Challenge Driven Innovation” (CDI) that helps to reformulate a task at hand into a series of modules or challenges that are addressed later either by a network of so-called *solvers* or internal company members. CDI have examples of different kind of challenges, including idea, validation, proof of concept, prototype, and production. So it is not specific to data labeling or algorithm development.

Seasonal events: there are many yearly data analysis events organized around the world. Usually, those are hosted by universities and attract quite a significant number of participants. International Data Analysis Olympiad (IDAO)^{qq} is just a single example among many.^{rr,ss} IDAO has engaged more than 2500 participants across 83 countries in 2020. Interesting and unique challenges might fit such events very well, and in that case, organizers will alleviate the burden of preparing and running the challenge quite significantly.

Other: There are many different venues for interactions between science and citizens. Michael Nielsen gives a good overview in his book “Reinventing Discovery: The New Era of Networked Science” [41]. A remarkable example of such interaction is the design of a network of micro-prediction agents that follow specific question-answering protocol. Authors of those agents get rewards for providing correct answers. Such protocol gives incentive to the participants to come up with better algorithms and suitable external data sources [42]. A broader list of citizen-science projects is, of course, available at Wikipedia [43].

^{PP}<https://innocentive.wazoku.com/>.

^{qq}<https://idao.world>.

^{rr}Data Mining Cup, <https://www.data-mining-cup.com/>.

^{ss}ASEAN Data Science Explorers<https://www.aseandse.org/>.

7. Open Datasets and Responsitories

Several datasets prepared for challenges are listed in the following. In addition, the LHC experiments have released some fraction of their data with corresponding simulated events (with ground truth) but there are no associated metrics. Also, authors of papers on the application of Machine Learning techniques to High Energy Physics are often willing to share their datasets on request, even if not formally released.

- HiggsML dataset is available on the CERN Open Data portal [6] with accompanying documentation [5]. All 818,238 events have been released including ground truth, while only a subset of 250,000 events is available on Kaggle.^{tt} For each event, it lists 17 low-level features and 13 high-level features for two classes. Beyond classification, a python script allows to introduce systematic effects [19, 21].
- Flavor of Physics challenge dataset.^{uu}
- Datasets for the TrackML challenge: (i) the Kaggle one [47] used for the Accuracy phase^{vv} (ii) the Codalab one [48] used for the Throughput phase.^{ww} Compared to the Accuracy one, a few features were corrected (iii) the CERN Open Data Portal final release in preparation.
- LHC Olympics-2020 dataset [39].
- LHCb Muon Identification challenge dataset^{xx} was published within International Data Analysis Olympiad-2019.^{yy}
- LHCb PID compression challenge,^{zz} with baseline solution.^{aaa}
- MiniBooNE Particle Identification dataset.^{bbb}

^{tt}<https://www.kaggle.com/c/higgs-boson>.

^{uu}<https://www.kaggle.com/c/flavours-of-physics/data>.

^{vv}<https://www.kaggle.com/c/trackml-particle-identification>.

^{ww}<https://competitions.codalab.org/competitions/20112>.

^{xx}<https://www.kaggle.com/kazeev/idao2019muonid>.

^{yy}<https://idao.world/history/#idao-2019>.

^{zz}<https://zenodo.org/record/1231531>.

^{aaa}https://github.com/weissercn/LHCb_PID_Compression.

^{bbb}<https://www.kaggle.com/ukveteran/miniboone-particle-identification>.

- LArTPC 2D/3D Simulation for Particle Segmentation & Clustering.^{ccc}
- Particle Identification from Detector Responses, a simplified dataset of a GEANT-based simulation of electron–proton inelastic scattering measured by a particle detector system.^{ddd}
- The top tagger dataset^{eee} has been used for extensive studies of top quark tagging [44].

There are several catalogs that reference HEP-related dataset, which can be handy for adding a published dataset to increase its visibility:

- CERN Open Data Portal^{fff} hosts a collection of datasets from all large LHC experiments as well as from OPERA experiment.
- The Durham High-Energy Physics Database.^{ggg} It hosts the data points from plots and tables related to several thousand publications including those from the LHC collaborations. It does not hold any event datasets unlike the CERN ODP.
- UCI ML HEP portal^{hhh} hosts a variety of HEP datasets associated with published papers, in particular the HIGGS UCI dataset [45] produced for the study [4] (see also Chapter 3).
- Inter-Experimental LHC Machine Learning (IML) Working Group datasets.ⁱⁱⁱ

8. Guidelines for New Competition Organizers

As for a movie or a novel, there are no rules which would guarantee the success of a scientific challenge. However, a set of guidelines can be derived from the experience gathered from the challenges summarized in this chapter, which does not pretend to be exhaustive.

^{ccc}<https://osf.io/vruzp/>.

^{ddd}<https://www.kaggle.com/naharrison/particle-identification-from-detector-responses>.

^{eee}Top tagging sample, <https://desycloud.desy.de/index.php/s/llbX3zpLhazgPJ6>, for more information and citation please use [46].

^{fff}<https://opendata.cern.ch>.

^{ggg}<https://www.hepdata.net/>.

^{hhh}<http://mlphysics.ics.uci.edu>.

ⁱⁱⁱ<https://iml.web.cern.ch/public-datasets>.

The overarching goal of a Machine Learning challenge is the scientific issue. It should be compelling both for experts of the domain (High-Energy Physicists), for experts in Machine Learning, and non-experts. It should be possible to pitch it to someone with little scientific background. At the same time, it should appear complex enough to be interesting, and non-physicists should feel they can contribute without a big disadvantage compared to physicists. It is important to focus on just one issue. For example, for an event classification problem like HiggsML or Flavor of Physics, one would provide particle 4-vectors and hide all the complexity of accurate calibration of the detector.

The centerpiece of a challenge is the dataset. It will have a life well beyond that of the challenge. In some sense, a challenge can be seen just as a way to advertise a dataset. The dataset should be curated and prepared to be easily understood and handled by non-experts, preferably with no need for non-standard tools. It should still retain some richness has the same dataset can be used later on for other challenges, tutorials, benchmarks. The preparation of the dataset is probably the more time-consuming part of a challenge preparation.

Since a challenge is by nature a competition, there should be a unique score to rank the participants. Domain experts are not used to ranking techniques based on a single number as they would typically like to see in-depth studies (with many curves and histograms) concerning various merit of a technique. Yet, there should be a single score, defined before the competition. Participants will optimize for this score, and the organizers bet that at the end of the competition the best algorithms from the point of view of the score, will also be the best algorithms for domain experts. Besides, the score should be sufficiently simple to be understood by the non-experts and stimulate their creativity (not a black box), robust against possible “hacks”, and, with limited luck factor when used in a challenge context. For example, in the HiggsML competition, the AMS (Eq. (1)) was chosen as it was much more relevant for a typical HEP classification problem (which are very unbalanced) than the usual ROC-AUC, and the regularization term 10 allowed to reduce the statistical uncertainty on the evaluation. Defining the score is probably the most difficult part of a challenge preparation.

Running a prototype of the challenge as part of an expert workshop or grad student school allows debugging many issues, from misleading documentation to the mechanics of the challenge platform.

Finally, challenges are a competitive market. Successful participants will spend months on a particular challenge, but they decide in little time in which competition they will enter. Without going into any details, this drives much of the effort in relying on an established challenge platform, on streamlining the challenge documentation (which should be readable by any scientific undergraduate and at the same time open up to more complex knowledge) and starting kit. In particular, submitting a first “hello world” solution should be possible in less than an hour. Public Relations is also important, as well as foreseeing incentives for participation. Money incentives are good but their role should not be over-emphasized. Invitations to participate in workshops at Machine Learning conferences or major HEP laboratories like CERN are valued by participants.

It can never be expected that the outcome of a challenge will be a piece of software ready to be plugged in. It is rather a smorgasbord of algorithms, well documented or not, forum posts or blogs.

Post-challenge workshops have the merit to keep some participants engaged in a collaboration with physicists (others will immediately move on to another competition). Special “jury” prizes set aside for algorithms judged on their overall merit (not the absolute best score, but also novelty, usability,...) allow keeping these participants engaged. Offering them the possibility to contribute to post-challenge papers is another means. The post-challenge phase is particularly interesting when it allows real collaboration, combining several good ideas, compared to the competition phase which is, well, a competition: discussion on the forum happens, notebooks are exchanged, but the best competitors are often silent until the end.

9. Conclusion

In this chapter, four quite different high-energy physics scientific competitions have been summarized. In all cases, new approaches have emerged, in addition to the optimization of existing ones. In all

cases, the formal end of the competition is actually the beginning of a new effort to sift through the wealth of information generated. Also, a long-lasting impact of the competitions is the dataset released with accompanying metric. High-energy physics boasts diversity and complexity of data structure and a variety of scientific questions raising interest well beyond its perimeter. It offers a wide range of future competition topics. Hopefully, resources, services and guidelines outlined in this chapter will help to pave the way for the design and organization of new fascinating challenges.

References

- [1] A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems 25*, eds. F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger (Curran Associates, Inc., 2012), pp. 1097–1105.
- [2] D. Donoho, 50 years of data science, in *Tukey Centennial Workshop* (2015).
- [3] A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, A. Himmel, A. Aurisano, K. Terao and T. Wongjirad, Machine learning at the energy and intensity frontiers of particle physics, *Nature* **560** (2018) 41.
- [4] P. Baldi, P. Sadowski and D. Whiteson, Searching for Exotic Particles in High-energy physics with deep learning, *Nature Commun.* **5** (2014) 4308; arXiv:1402.4735 [hep-ph].
- [5] C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl and D. Rousseau, Learning to discover: the Higgs boson machine learning challenge — documentation, CERN Open Data Portal (2014); doi:10.7483/OPENDATA.ATLAS.MQ5J.GHXA.
- [6] ATLAS Collaboration, Dataset from the ATLAS Higgs boson machine learning challenge 2014, CERN Open Data Portal (2014); doi:10.7483/OPENDATA.ATLAS.ZBP2.M5T8.
- [7] C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl and D. Rousseau, The Higgs Boson Machine Learning Challenge (PMLR, Montreal, Canada, 2015).
- [8] G. Cowan, C. Germain, I. Guyon, B. Kégl and D. Rousseau (eds.), *Proceedings of Machine Learning Research*, Vol. 42 (PMLR, 2014).
- [9] ATLAS, G. Aad *et al.*, Evidence for the Higgs-boson Yukawa coupling to tau leptons with the ATLAS detector, *J. High Energy Phys.* **04** (2015) 117; arXiv:1501.04943 [hep-ex].
- [10] G. Cowan, K. Cranmer, E. Gross and O. Vitells, Asymptotic formulae for likelihood-based tests of new physics, *Eur. Phys. J. C* **71** (2011) 1554; arXiv:1007.1727 [physics.data-an].

- [11] T. Chen and C. Guestrin, XGBoost: A scalable tree boosting system, in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, KDD '16* (ACM, New York, NY, 2016); doi:10.1145/2939672.2939785.
- [12] F. Chollet. <https://twitter.com/fchollet/status/1113476428249464833> (2019).
- [13] ATLAS, M. Aaboud *et al.*, Evidence for the associated production of the Higgs boson and a top quark pair with the ATLAS detector, *Phys. Rev. D* **97** (2018) 072003; arXiv:1712.08891 [hep-ex].
- [14] L. Polson and A. Scaife, Introduction to machine learning (2020); <https://hsf-training.github.io/hsf-training-ml-webpage/>.
- [15] O. Schwander, Fouille de données et médias sociaux. tp1: Higgs boson machine learning challenge (2017); http://dac.lip6.fr/master/wp-content/uploads/2018/09/fdms-2018-2019_tp1.pdf (in French).
- [16] J. Wittenauer, Lessons learned from the higgs boson kaggle challenge, (2014); <https://towardsdatascience.com/lessons-learned-from-the-higgs-boson-kaggle-challenge-537b50e8c899>.
- [17] P. Chakravarti, Inference for clustering and anomaly detection, PhD thesis, Department of Statistics & Data Science, Carnegie Mellon University, (2020).
- [18] G. Chatham Strong, On the impact of selected modern deep-learning techniques to the performance and celerity of classification models in an experimental high-energy physics use case, *Mach. Learning Sci. Technol.* **1** (2020) 045006; arXiv:2002.01427 [physics].
- [19] V. Estrade, Victor-estrade/datawarehouse: First release (2018); doi:10.5281/zenodo.1887847.
- [20] V. Estrade, C. Germain, I. Guyon and D. Rousseau, Adversarial learning to eliminate systematic errors: a case study in high energy physics, in *Proc. Deep Learning for Physical Sciences Workshop at NIPS* (2017).
- [21] V. Estrade, PhD thesis, Université Paris-Saclay, Sciences et Technologies de l'Information et de la Communication (2021), In preparation.
- [22] S. Wunsch, S. Jörgen, R. Wolf and G. Quast, Reducing the dependence of the neural network function to systematic uncertainties in the input space, *Comput. Soft. Big Sci.* **4** (2020).
- [23] S. Wunsch, S. Jörgen, R. Wolf and G. Quast, Optimal statistical inference in the presence of systematic uncertainties using neural network optimization based on binned Poisson likelihoods with nuisance parameters (2020); arXiv:2003.07186 [physics.data-an].
- [24] R. Aaij *et al.*, Search for the lepton flavour violating decay $\tau^- \rightarrow \mu^- \mu^+ \mu^-$, *J. High Energy Phys.* **2015** (2015) 121.
- [25] H. Cramr, On the composition of elementary errors, *Scandinavian Actuarial J.* **1** (1928) 141.
- [26] A. L. Read, Presentation of search results: the CL_S technique, *J. Phys. G* **28** (2002) 2693.
- [27] Heavy Flavor Averaging Group (HFAG), Y. Amhis *et al.*, Averages of b -hadron, c -hadron, and τ -lepton properties as of summer 2014 (2014); arXiv:1412.7515 [hep-ex].

- [28] S. Amrouche, L. Basara, P. Calafiura, V. Estrade, S. Farrell, D. R. Ferreira, L. Finnie, N. Finnie, C. Germain, V. V. Gligorov, T. Golling, S. Gorbunov, H. Gray, I. Guyon, M. Hushchyn, V. Innocente, M. Kiehn, E. Moyse, J.-F. Puget, Y. Reina, D. Rousseau, A. Salzburger, A. Ustyuzhanin, J.-R. Vlimant, J. S. Wind, T. Xylouris and Y. Yilmaz, The tracking machine learning challenge: Accuracy phase, in *The NeurIPS 2018 Competition* (Springer, 2019), pp. 231–264.
- [29] S. Amrouche, L. Basara, P. Calafiura, D. Emelyanov, V. Estrade, S. Farrell, C. Germain, V. V. Gligorov, T. Golling, S. Gorbunov, H. Gray, I. Guyon, M. Hushchyn, V. Innocente, M. Kiehn, M. Kunze, E. Moyse, D. Rousseau, A. Salzburger, A. Ustyuzhanin, J.-R. Vlimant and Y. Yilmaz, The tracking machine learning challenge: Throughput phase; arXiv:2105.01160 [cs.LG].
- [30] S. Amrouche *et al.*, Track reconstruction at LHC as a collaborative data challenge use case with RAMP, *EPJ Web Conf.* **150** (2017) 00015.
- [31] D. Rousseau, Connecting the dots in high-energy physics, *Nature Machine Intell.* **1** (2019) 288.
- [32] ATLAS, C. Gumpert, A. Salzburger, M. Kiehn, J. Hrdinka and N. Calace, ACTS: from ATLAS software towards a common track reconstruction software, *J. Phys. Conf. Ser.* **898** (2017) 042011.
- [33] A. Zlokapa, A. Anand, J.-R. Vlimant, J. M. Duarte, J. Job, D. Lidar and M. Spiropulu, Charged particle tracking with quantum annealing-inspired optimization (2019); arXiv:1908.04475 [quant-ph].
- [34] X. Ju *et al.*, Graph neural networks for particle reconstruction in high energy physics detectors, in *33rd Annual Conf. Neural Information Processing Systems* (2020); arXiv:2003.11603 [physics.ins-det].
- [35] X. Wang, L. Besançon, D. Rousseau, M. Sereno, M. Ammi and T. Isenberg, Towards an understanding of augmented reality extensions for existing 3D data analysis tools, in *Proc. 2020 CHI Conf. Human Factors in Computing Systems* (ACM, 2020).
- [36] X. Wang, Augmented reality environments for the interactive exploration of 3D data. Ph.D. thesis, Université Paris-Saclay Sciences et Technologies de l'Information et de la Communication (2020).
- [37] M. Cacciari, G. P. Salam and G. Soyez, The anti- k_t jet clustering algorithm, *J. High Energy Phys.* **2008** (2008) 063.
- [38] K. Agashe, P. Du, S. Hong and R. Sundrum, Flavor universal resonances and warped gravity, *J. High Energy Phys.* **2017** (2017).
- [39] G. Kasieczka, B. Nachman and D. Shih, Official datasets for LHC olympics 2020 anomaly detection challenge, (2019); doi:10.5281/zenodo.3596919.
- [40] D. Clery, Galaxy zoo volunteers share pain and glory of research, *Science* (2011).
- [41] M. Nielsen, *Reinventing Discovery: The New Era of Networked Science*, Vol. 70 (Princeton University Press, 2020).
- [42] P. Cotton, Self organizing supply chains for micro-prediction: Present and future uses of the roar protocol (2019); arXiv:1907.07514 [stat.AP].
- [43] List of crowdsourcing projects, https://en.wikipedia.org/wiki/List_of_crowdsourcing_projects.

- [44] A. Butter *et al.*, The machine learning landscape of top taggers, *SciPost Phys.* **7** (2019) 014; arXiv:1902.09914 [hep-ph].
- [45] P. Baldi, P. Sadowski and D. Whiteson, Higgs dataset (2014); <https://archive.ics.uci.edu/ml/datasets/HIGGS>.
- [46] A. Butter, G. Kasieczka, T. Plehn and M. Russell, Deep-learned top tagging with a Lorentz layer, *SciPost Phys.* **5** (2018) 028; arXiv:1707.08966 [hep-ph].
- [47] A. Salzburger, V. Innocente, J.-R. Vlimant, D. Rousseau, V. Gligorov, V. Estrade, L. Basara, P. Calafiura, S. Farell, H. Gray, T. Golling, M. Kiehn, S. Amrouche, A. Ustyuzhanin, M. Hushchyn, E. Moyse, C. Germain, and I. Guyon, TrackML particle tracking challenge (2018); doi:10.5281/zenodo.4730167.
- [48] A. Salzburger, V. Innocente, J.-R. Vlimant, D. Rousseau, V. Gligorov, L. Basara, V. Estrade, P. Calafiura, S. Farell, H. Gray, T. Golling, M. Kiehn, S. Amrouche, M. Hushchyn, A. Ustyuzhanin, E. Moyse, C. Germain, and I. Guyon, TrackML throughput phase (2018); doi:10.5281/zenodo.4730167.

Index

- ABCNet, 415
- AdaBoost, 36
- ADADELTA, 323, 744
- Adam, 323, 744
- adjacency matrix, 392
- AlexNet, 319, 326
- allreduce, 250
- anomaly detection, 488
- anti- k_t algorithm, 547
- approximate median significance (AMS), 16, 768
- arc-lengths, 750
- architecture, 725
- area under the curve (AUC), 15
- artificial neural network, 318
- ATLAS experiment, 466

- backpropagation through time, 544
- backward elimination, 31
- bagging, 53
- Bayesian optimization, 257, 743
- bias, 18, 35, 754
- bias-variance trade-off, 18, 20
- bias-variance ratio, 754
- bidirectional recurrent neural networks (BRNNs), 544
- boosted jet, 445
- boosted decision trees (BDT), 34, 314
- boosted Higgs boson tagging, 466

- calorimeter deposits (clusters), 557
- Cambridge–Aachen jets, 559
- CatBoost, 54
- closure tests, 731
- clustering, 355
- clustering history, 560
- clustering tree, 562
- CMS experiment, 466
- CMS Open Data, 336
- combinatorial Kalman filter, 408
- complex cells, 319
- computer vision, 318, 439
- convolution neural network (CNN), 325, 326, 347, 440, 569
- convolutional layers, 320, 325, 331, 340, 343–345, 544
- correlation coefficient, 29, 30, 726
- cross-validation, 12, 256, 741, 770

- data augmentation, 52, 73, 463, 772
- data distributed training, 250
- decision trees, 20
- decorrelated tagging, 463
- deep sets, 399, 571
- DeepCSV, 553
- DeepFlavor, 553
- dense neural network, 770
- DenseNet, 337, 338
- DIPS, 572
- distributed training, 243
- dynamic graph CNN, 398

- edge convolution, 396
- EdgeConv, 395
- effective number of events, 25
- ensemble learning, 33
- equivariant, 320
- evolutionary algorithms, 257
- exascale, 260

- F1 metric, 334
- Faster R-CNN, 326
- FastKernel, 727
- feature extractor, 344, 346
- feature maps, 320, 325, 343, 344
- figure of merit, 14, 38, 768, 777
- fine tuning, 334
- forward greedy selection, 31
- fully connected layer, 329, 331, 340
- functional uncertainty, 734
- future testing, 751

- GAPNet, 415
- GarNet, 399
- gated recurrent units (GRUs), 544, 545
- generative models, 487
- genetic algorithms, 727
- geometric deep learning, 388
- GoogLeNet, 325
- GRACLU, 402
- gradient boosting, 41
- gradient descent, 738
- graph network formalism, 396
- graph neural network (GNN), 387, 388
- GravNet, 395

- heavy flavor jets, 547
- HL-LHC, 759
- Hough transform, 408
- hybrid parallelism, 258
- hypercomplex cells, 319
- hyperopt, 743
- hyperparameter, 24, 322, 323, 738
- hyperparameter optimization, 247

- image representation, 439
- ImageNet, 319, 334
- impact parameter, 549
- instance segmentation, 368
- interpolation regime, 20
- interpolation uncertainty, 734
- interpretability, 480

- jet clustering history, 562
- jet energy regression, 485
- jet image, 439
- jet tagging, 454

- k -folding, 752
- k -nearest neighbors (kNN), 395
- Keras, 738
- Kullback–Leibler divergence, 346

- Large Hadron Collider (LHC), 440
- learning rate, 323, 334
- LightGBM, 54
- locality sensitive hashing, 422
- locally connected network, 337
- long short-term memory (LSTM), 544, 545, 551
- look-back, 730
- loss function, 18, 42, 322, 323, 741

- machine learning particle flow (MLPF), 421
- max pooling, 321
- max pooling layer, 331, 340, 344
- mean decrease accuracy (MDA), 30
- mean decrease impurity (MDI), 29
- message-passing, 396
- model parallelism, 253
- momentum fraction, 725
- MoNet, 341
- Monte Carlo, 11, 12, 28, 52, 722
- multi-head attention, 400
- multilayer perceptron (MLP), 318–320, 331, 337, 338

- naïve Bayes, 550
- natural language processing (NLP), 541
- neural networks, 720
- NNPDF, 716
- nodal mutation, 729
- nuisance parameters, 772
- object condensation, 420
- occlusion test, 343, 348, 349
- optimizers, 322, 323
- overtraining, 10, 38, 45
- parallel computations, 258
- parameter distribution, 248
- particle-flow (PF) reconstruction, 419
- parton, 715
- parton distribution function (PDF), 715
- patience, 741
- pattern recognition, 720
- permutation importance, 30
- pileup removal, 485
- point change mutations, 728
- PointNet, 399
- pooling layers, 320, 321, 344
- precision, 334
- preprocessing, 725
- pruning, 31
- QCD, 718
- quark vs. gluon jet discrimination, 568
- quark vs. gluon tagging, 467
- random forests, 53
- recall, 334
- receiver operating characteristic (ROC) curve, 14
- receptive fields, 319
- recurrent neural network, 338, 543
- recursive neural networks (RecNNs), 565
- regression trees, 47
- regularization, 43, 75, 277, 371, 463
- ResNet, 327, 336
- RMSProp, 323, 744
- RNNIP, 551
- Scikit-learn, 54
- SE-ResNet-34, 328
- secondary vertex, 548, 555
- self-attention, 399
- self-attention graph (SAG), 402
- semantic segmentation, 359
- sequence, 541
- sequence-based learning, 541
- sequential clustering algorithms, 547
- Set2Graph, 413
- sigmoid, 725
- significance, 16
- simple cells, 319, 320
- softmax activation, 340
- softmax function, 344
- staleness of gradients, 248
- stochastic gradient descent, 323
- stopping, 729
- strange jets, 554
- surrogate splits, 29
- systematic uncertainties, 49, 597, 614
- t*-distributed stochastic neighbor embedding (*t*-SNE), 346–348
- tau lepton jets, 556
- TensorFlow, 738
- testing set, 12, 17, 18, 38–40, 45, 46, 747
- TMVA, 54
- top jets, 559
- top tagging, 458
- track reconstruction, 407
- tracking, 407
- TrackML, 409
- training, 730
- transfer learning, 333–335
- transformers, 399, 571
- trimming, 560
- uncertainties, 471

- validation, 720, 730
- vanishing or exploding gradient
 - problem, 545
- variable ranking, 29
- variance, 18, 32, 53, 754
- VGG model, 334
- visual cortex, 319, 320
- W/Z Tagging, 455
- XGBoost, 54